

**MVME117**  
**Firmware Debug Monitor**  
**User's Manual**



***MICROSYSTEMS***

**QUALITY • PEOPLE • PERFORMANCE**

**MVME117 FIRMWARE DEBUG MONITOR  
USER'S MANUAL**

The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. Furthermore, Motorola reserves the right to make changes to any products herein to improve reliability, function, or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights or the rights of others.

EXORmacs, I/Omodule, MVME117bug, SYSTEM V/68, and VERSAdos are trademarks of Motorola Inc.

SASI is a trademark of Shugart Associates.

The computer program stored in the Read Only Memory of this device contains material copyrighted by Motorola Inc., first published 1985, and may be used only under a license such as the License For Computer Programs (Article 14) contained in Motorola's Terms and Conditions of Sale, Rev. 1/79.

First Edition

Copyright 1986 by Motorola Inc.



## TABLE OF CONTENTS

	<u>Page</u>
<b>CHAPTER 1</b>	<b>GENERAL INFORMATION</b>
1.1	INTRODUCTION ..... 1
1.2	DEFINITION OF MVME117 FIRMWARE DEBUG MONITOR ..... 1
1.3	FIRMWARE MONITOR INTERNAL STRUCTURE ..... 1
1.3.1	Memory Map ..... 1
1.3.2	Vectors and Errors ..... 2
1.3.2.1	Resetting Vector Base Register ..... 3
1.3.3	Disk I/O ..... 4
1.3.3.1	SCSI Disk I/O ..... 5
1.3.3.2	RWIN Disk I/O ..... 5
1.4	RELATED DOCUMENTATION ..... 6
 <b>CHAPTER 2</b>	 <b>MVME117 FIRMWARE DEBUG MONITOR OPERATING PROCEDURE</b>
2.1	INTRODUCTION ..... 7
2.2	MVME117 SWITCHES ..... 7
2.2.1	RESET ..... 7
2.2.2	Software ABORT ..... 9
2.3	POWER-UP SEQUENCE ..... 10
2.3.1	Power-Up Test Enabled ..... 10
2.3.2	ROMBOOT Code Installed ..... 10
2.3.3	Autoboot Enabled ..... 10
2.3.4	Normal Processing at Power-Up ..... 11
2.3.5	First Time Power-Up ..... 11
2.4	TERMINAL CONTROL CHARACTERS ..... 12
2.5	COMMAND PROCESSING ..... 12
2.6	ROMBOOT FACILITY ..... 14
2.6.1	Autoboot Support ..... 17
2.6.2	Power-Up Test ..... 17
 <b>CHAPTER 3</b>	 <b>COMMAND LINE FORMAT</b>
3.1	INTRODUCTION ..... 19
3.2	FIRMWARE MONITOR COMMAND LINE FORMAT ..... 20
3.2.1	Expression as a Parameter ..... 20
3.2.2	Address as a Parameter ..... 20
3.2.2.1	Address Formats ..... 21
3.2.2.2	Offset Registers ..... 21
3.3	COMMAND VERIFICATION ..... 22

## TABLE OF CONTENTS (cont'd)

	<u>Page</u>
CHAPTER 4	COMMAND SET
4.1	INTRODUCTION ..... 23
4.2	FIRMWARE MONITOR COMMANDS ..... 24
4.2.1	Display/Set Register (<register>) ..... 25
4.2.2	Autoboot Enable/Disable (AB and NOAB) ..... 26
4.2.3	Boot Dump (BD) ..... 29
4.2.4	Block Fill (BF) ..... 31
4.2.5	Boot Halt (BH) ..... 32
4.2.6	Block Initialize (BI)..... 33
4.2.7	Block Move (BM) ..... 34
4.2.8	Boot Load (BO) ..... 35
4.2.9	Breakpoint Set and Remove (BR and NOBR) ..... 38
4.2.10	Block of Memory Search (BS) ..... 40
4.2.11	Block of Memory Test (BT) ..... 42
4.2.12	Display CMOS RAM Variables (CMOS) ..... 43
4.2.13	Calculate Checksum (CS) ..... 44
4.2.14	Data Conversion (DC) ..... 48
4.2.15	Display Formatted Registers (DF) ..... 49
4.2.16	Dump Memory (S-Records) (DU) ..... 51
4.2.17	Go Direct Execute Program (GD) ..... 53
4.2.18	Go Execute Program (GO) ..... 54
4.2.19	Go Until Breakpoint (GT) ..... 56
4.2.20	Help (HE) ..... 58
4.2.21	I/O Command for Drive/Controller (IOC) ..... 59
4.2.22	I/O Physical for Drive/Controller (IOP) ..... 65
4.2.23	I/O Teach for a Drive/Controller (IOT) ..... 68
4.2.24	Load (S-Records) (LO) ..... 72
4.2.25	Memory Display (MD) ..... 75
4.2.26	Memory Modify (MM) ..... 78
4.2.27	Memory Set (MS) ..... 81
4.2.28	Display Offsets (OF) ..... 83
4.2.29	Printer Attach and Detach (PA and NOPA) ..... 85
4.2.30	Port Format (PF) ..... 87
4.2.31	Power-Up Test Enable/Disable (PT and NOPT) ..... 92
4.2.32	SCSI/SASI Bus Reset or SCSI Controller Reset (RESET) .... 93
4.2.33	Set Date and Time of Day (SET) ..... 94
4.2.34	Self-Test (ST) ..... 95
4.2.35	Terminal Attach (TA) ..... 100
4.2.36	Display Current Date and Time of Day (TIME) ..... 101
4.2.37	Transparent Mode (TM) ..... 102
4.2.38	Trace (TR) ..... 105
4.2.39	Trace to Temporary Breakpoint (TT) ..... 107
4.2.40	Verify (S-Records) (VE) ..... 108
4.2.41	Vector Initialize (VI) ..... 111
4.3	COMMAND SUMMARY ..... 112

**TABLE OF CONTENTS (cont'd)**

	<u>Page</u>
 CHAPTER 5      USING THE ASSEMBLER/DISASSEMBLER	
5.1      INTRODUCTION .....	115
5.1.1      M68010 Assembly Language .....	115
5.1.1.1      Machine-Instruction Operation Codes .....	115
5.1.1.2      Directives .....	115
5.1.2      Comparison with MC68000 Resident Structured Assembler ...	116
5.2      SOURCE PROGRAM CODING .....	117
5.2.1      Source Line Format .....	117
5.2.1.1      Operation Field .....	117
5.2.1.2      Operand Field .....	118
5.2.1.3      Disassembled Source Line .....	118
5.2.1.4      Mnemonics and Delimiters .....	119
5.2.1.5      Character Set .....	120
5.2.2      Instruction Summary .....	121
5.3      ENTERING AND MODIFYING SOURCE PROGRAMS .....	121
5.3.1      Invoking the Assembler/Disassembler .....	123
5.3.2      Entering a Source Line .....	123
5.3.3      Program Entry/Branch and Jump Addresses .....	124
5.3.3.1      Entering Absolute Addresses .....	124
5.3.3.2      Desired Instruction Form .....	125
5.3.3.3      Current Location .....	125
5.3.4      Assembler Output/Program Listings .....	125
5.3.5      Error Conditions and Messages .....	126
5.3.5.1      Error Traps .....	126
5.3.5.2      Improper Character .....	127
5.3.5.3      Number Too Large .....	128
5.3.5.4      Assembly Errors .....	128
 CHAPTER 6      MVME117 FIRMWARE MONITOR ROUTINES	
6.1      INTRODUCTION .....	131
6.2      USER I/O THROUGH TRAP #15 .....	131
6.2.1      TRAP #15 Summary .....	132
6.2.2      TRAP #15 User Interface .....	132
6.2.2.1      Normal TRAP #15 Calls .....	132
6.2.2.2      MVME117 Firmware Port Assignments .....	135
6.2.2.3      SCSI TRAP #15 Calls .....	135
6.2.2.4      Example TRAP #15 Program .....	138
6.3      MVME117 FIRMWARE MONITOR SUBROUTINES .....	140
 APPENDIX A      SOFTWARE ABORT .....	
APPENDIX B      FIRMWARE MONITOR MESSAGES.....	145
APPENDIX C      CONFIGURATION AREA .....	149
APPENDIX D      S-RECORD OUTPUT FORMAT .....	151

## TABLE OF CONTENTS (cont'd)

		<u>Page</u>
APPENDIX E	SCSI FIRMWARE SUPPORT	
E.1	INTRODUCTION .....	157
E.2	MODES OF OPERATION .....	157
E.3	MVME117 SCSI FIRMWARE ENTRY POINTS .....	157
E.4	EQUIPMENT SUPPORTED .....	159
E.5	EXPLAINING PACKETS .....	160
E.5.1	Read/Write Packet .....	160
E.5.2	Attach/Detach Packet .....	162
E.5.3	Format Packet .....	166
E.5.4	Assign Alternate Sector Packet (SCSI) .....	166
E.5.5	Assign Alternate Track Packet (SASI) .....	169
E.5.6	Custom SCSI Sequence Packet .....	171
E.5.7	SCSI Bus Reset Packet .....	175
E.5.8	SCSI Controller Reset Packet .....	175
E.6	CUSTOM SCSI SEQUENCE DETAILS .....	177
E.7	PACKET CHECKING .....	181
E.8	SCSI FIRMWARE INTERRUPT STRUCTURE .....	186
E.9	WRITING A DRIVER .....	188
E.9.1	Introduction .....	191
E.9.2	Building the Packet .....	191
E.9.3	Passing Commands to the SCSI Firmware .....	191
E.9.4	Return to the Driver from SCSI Firmware .....	192
E.9.4.1	Final Status .....	192
E.9.4.2	Intermediate Status .....	194
INDEX	.....	197

## LIST OF ILLUSTRATIONS

FIGURE	2-1. Power-Up/Reset .....	8
	2-2. Switch Location .....	9
	2-3. MVME117 Command Processing .....	13
	5-1. Sample Program to Convert ASCII Digit to Hexadecimal Value .....	122
	5-2. ASCII Character Set .....	122
	5-3. Sample Program as Entered into MVME117 .....	124
	5-4. Sample Program Listing .....	126
	5-5. Examples of Error Traps .....	127
	5-6. Examples of Improper Characters .....	128
	5-7. Example of a Number Which Is Too Large .....	128
	5-8. Examples of Assembly Errors .....	129

## LIST OF TABLES

TABLE	4-1. Firmware Monitor Commands by Type .....	23
	4-2. Firmware Monitor Command and Option Summary .....	112

## CHAPTER 1

### GENERAL INFORMATION

#### 1.1 INTRODUCTION

This manual describes firmware debug monitor as it is used in the MVME117 Monoboard Microcomputer, hereafter referred to as the MVME117.

#### 1.2 DEFINITION OF THE MVME117 FIRMWARE DEBUG MONITOR

This product is the resident firmware debugging package for the MVME117. The 64Kb firmware (stored in EPROM devices) provides a self-contained programming and operating environment. This firmware monitor interacts with the user through predefined commands that are entered via the terminal. The commands fall into four general categories:

- a. Commands which allow the user to display or modify memory.
- b. Commands which allow the user to display or modify the various internal registers of the MC68010.
- c. Commands which allow the user to execute a program under various levels of control.
- d. Commands which control access to the various input/output resources on the board.

An additional function called the TRAP #15 handler allows the user program to utilize various routines within the firmware monitor. The TRAP #15 handler is discussed in Chapter 6.

The operational mode of the firmware monitor is described in Chapter 2.

#### 1.3 FIRMWARE MONITOR INTERNAL STRUCTURE

##### 1.3.1 Memory Map

The following abbreviated memory map for the MVME117 highlights addresses that might be of particular interest to the firmware monitor user. Note that addresses are assumed to be hexadecimal throughout this manual. In text, numbers may be preceded with a dollar sign (\$) for identification as hexadecimal.

<u>RAM LOCATION</u>	<u>FUNCTION</u>
0-3FF	Vectors
400-1FFF	Work area and stack for the firmware monitor

<u>SPECIAL LOCATIONS</u>	<u>FUNCTION</u>
F00000-F00007	Area containing initial values for supervisor stack pointer and program counter. Refer to paragraph 2.2.1.
F40F21-F40FEF(with 2K x 8 CMOS RAM)	CMOS RAM used to save SCSI and other debug monitor information (Autoboot, etc.). Refer to NOTES on the
F43F21-F43FEF(with 8K x 8 CMOS RAM)	Autoboot (AB) command in paragraph 4.2.2 for additional discussion.
F48005	local serial port #1
F48001	local serial port #2
F44001	local printer port
FFE0D1	(optional) RWIN disk controller
FFE1C9	(optional) MVME400 serial port #1 (7201/A)
FFE1CB	(optional) MVME400 serial port #2 (7201/B)
FFE1E1	(optional) MVME410 printer port #1 (PIA/A)
FFE1E9	(optional) MVME410 printer port #2 (PIA/B)
FF1000	(optional) MVME050 serial port #1
FF1040	(optional) MVME050 serial port #2
FF1080	(optional) MVME050 printer port

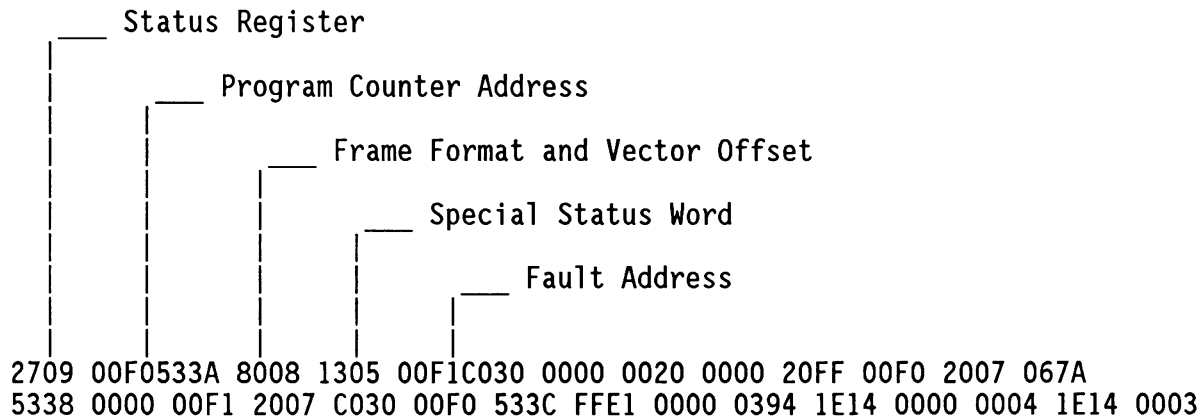
### 1.3.2 Vectors and Errors

The firmware monitor shares resources with the target program under test -- that is, each affected resource can be used only by the monitor or the target program at any given time.

The firmware exception vectors are memory locations from which the processor fetches the address of a routine which will handle the exception. These vectors are initialized by the firmware monitor in default memory locations 0 through \$3FF during a cold start sequence (refer to Chapter 2). If the target program uses any of these locations, the user values must be rewritten following each cold start. If the target program uses any of the following locations, the associated function will be lost to the firmware monitor.

<u>MEMORY LOCATION</u>	<u>FIRMWARE MONITOR FUNCTION</u>
8-B	Memory sizing, general bus error handling
10-13	Breakpoints (illegal instructions)
24-27	Trace
BC-BF	TRAP #15 user calls to firmware monitor
7C-7F	Software ABORT switch (refer to Appendix A)
110-113	Vector #44 to SCSI I/O
118-11B	Vector #46 from SCSI I/O

When uninitialized vectors are given control because of exception processing, the console terminal displays a general message indicating the vector offset that was used: Offset Vector \$xxx Error Trap. In addition, several of the vectors cause display of appropriate information. (Refer to Appendix B for a list of error messages.) BUS and ADDR error traps also cause display of the exception status from the stack, in hexadecimal characters, as shown in the following example.



#### BUS ERROR TRAP

For additional information on this display, refer to the bus error, address error, and the reference classification descriptions in the exception processing chapter of the M68000 16/32-Bit Microprocessor Programmer's Reference Manual.

**1.3.2.1 Resetting Vector Base Register.** The MC68010 processor upon which the MVME117 is based features a Vector Base Register (VBR) which contains the base (starting) address for the MVME117 exception vectors. Exception vectors are located in memory addresses 0 through \$3FF relative to the VBR. Upon reset (cold or warm start) of the MC68010, the value of the VBR is set to zero.

The firmware monitor must have control of the exception vectors to function properly. If the user sets the VBR to a value other than its default value of zero, the user must also establish a new set of exception vector memory locations for the VBR value. In other words, the user must copy all existing vector memory locations to the same relative location in the new VBR table.

In the following example, the VBR value is changed from 0 to 10F00. Exception vector memory locations must also be copied to this new location. Note that the content of each vector memory location (i.e., the appropriate routine address) remains the same.

VBR = 0		VBR = 10F00	
0	00000444	10F00	00000444
4	0000044C	10F04	0000044C
8	00000454	10F08	00000454
C		10F0C	
//		//	
//		//	
3FC	000008A4	112FC	000008A4

### 1.3.3 Disk I/O

The firmware monitor provides "Polled Mode" disk I/O support through either an onboard Small Computer Systems Interface (SCSI) bus controller, or an optional Winchester disk controller (RWINx) connected through a VMEbus to I/O Channel interface module (MVME316). I/O requests handled by the common debugger disk routines require "Starting Block Number" and "Number of Blocks" along with other information (controller, drive, I/O area, etc.). Of particular interest is that the "Blocks" just mentioned are always 256-byte blocks (logical sectors), and not physical sectors. Any conversions required to allow sector sizes other than 256 bytes are done internally within the debug monitor firmware. The user interface with disk I/O through the debug monitor is usually via the B0, BH, BD, IOT, and IOP commands. These all make use of VERSAdos 256-byte blocks as they perform their specific functions. The IOC command, however, actually allows an unconverted I/O request to be made.

By using the VERSAdos format, specific variations in media support can be handled, for the most part, without intervention by the user. The B0, BH, BD, IOP, and optionally the IOT command all read the volume ID information contained within the first 256 bytes of the media, previously initialized. Contained in this area are pointers to the VERSAdos configuration data. The configuration data is used to set up the appropriate conversion for 256-byte block to physical sector number and to configure the controller supporting the drive being used. The volume ID is recognized by locating a semaphore containing "MOTOROLA" or "EXORMACS" in positions \$F8 through \$FF. A complete description of the volume ID follows:

<u>BYTES</u>	<u>USED FOR</u>
\$14-\$17	Starting sector address of program to be loaded (via BH, B0).
\$18-\$19	Number of 256-byte sectors to be loaded.
\$1E-\$21	Load address (first destination memory byte).
\$40-\$7F	64-byte test pattern used by the "ST" command.
\$90-\$93	Sector address of media configuration parameters (refer to Appendix C).
\$94	Length of configuration area (usually one 256-byte block).
\$F8-\$FF	Special code. (Refer to preceding paragraph.)

**1.3.3.1 SCSI Disk I/O.** The primary disk support provided is through an NCR5380 SCSI chip located on the MVME117. This controller accommodates the SCSI interface as defined by the ANSI X3T9.2 Committee (Revision 14 of the SCSI specification).

The controller supports arbitration (of the SCSI bus) and reselection. It can operate in both the "Initiator" and "Target" modes and runs in the single-ended, open collector SCSI bus implementation.

Up to seven other SCSI controllers can be connected to the bus, each able to control a theoretical maximum of 8 devices. The disk commands supported in the firmware monitor assume SCSI controllers if the controller number provided is 0 through 7. The controller number is used as the level ID on the SCSI bus. The MVME117 is assigned by default as controller level ID 7 but can be reassigned with the IOT command.

For more specific information about setting up I/O requests through the SCSI bus, refer to Chapter 6 and Appendix E.

**1.3.3.2 RWIN Disk I/O.** Support for Winchester and floppy disk via an M68RWIN controller is provided within the firmware monitor primarily as an internal Motorola development tool. To use the RWIN controller, an I/O channel interface module (MVME316) is required.

Device assignments for the RWIN controller are fixed, with drives 0 and 1, if present, being hard disks. Drives 2 and 3, if present, are for floppy media.

The disk commands assume RWIN if controller number 8 is entered.

If a disk read to a Winchester drive fails, the read is retried twice before generating an error message. The RWIN controller makes corrections, if possible, and the disk transfer continues on from the next sector.

For information on interpreting data displayed for error conditions, refer to the Winchester Disk Controller User's Manual as listed in paragraph 1.4.

#### 1.4 RELATED DOCUMENTATION

The following publications may provide additional helpful information. If not shipped with this product, they may be obtained from Motorola's Literature Distribution Center, 616 West 24th Street, Tempe, AZ 85282; telephone (602) 994-6561. Non-Motorola documents may be obtained from the sources listed.

DOCUMENT TITLE	MOTOROLA PUBLICATION NUMBER
MVME117 VMEmodule Monoboard Microcomputer and MVME708-1 Transition Module User's Manual	MVME117
VERSAdos to VME Hardware and Software Configuration User's Manual	MVMEVDOS
Winchester Disk Controller User's Manual	M68RWIN1
MVME400 Dual RS-232C Serial Port Module User's Manual	MVME400
MVME410 Dual Parallel Port Module User's Manual	MVME410
M68000 16/32-Bit Microprocessor Programmer's Reference Manual	M68000UM
MVME050 System Controller Module and MVME701/MVME701A I/O Transition Module User's Manual	MVME050
M68000 Family VERSAdos System Facilities Reference Manual	M68KVSF
=====	
Saber - 55M Disk Link Technical Manual; 950-000023; Adaptive Data Systems Inc., 2627 Pomona Blvd., Pomona, California 91768	
DTC-520B Controller Specification; 09-00143 Rev 01; Data Technology Corporation, 2775 Northwestern Parkway, Santa Clara, CA 95051	
SCSI Small Computer System Interface; draft X3T9.2/82-2 - Revision 14; Computer and business Equipment Manufacturers Association, 311 First Street, N.W., Suite 500, Washington, D.C. 20001	

**CHAPTER 2****MVME117 FIRMWARE DEBUG MONITOR OPERATING PROCEDURE****2.1 INTRODUCTION**

The following procedures enable the user to enter the MVME117 firmware debug monitor. For information on hardware configuration, especially for EPROM size and speed, refer to the MVME117 VMEmodule Monoboard Microcomputer User's Manual.

**2.2 MVME117 SWITCHES****2.2.1 RESET**

When this momentary-action switch is pressed, it resets the MVME117 logic circuits and maps the ROM to RAM address \$000000 for four read cycles where the processor extracts the stack pointer (\$000000) and program counter (\$000004). Following this, the ROM is remapped back to its regular \$F00000 address. Execution within the debug monitor begins by examining the cold start bit in the status register in the MVME117 to see if this is the first time the module has come through "RESET" (see Figure 2-1). If so, test and diagnostic routines may be invoked along with memory initialization routines. If not, the following vectors are checked to see if they are initialized properly for successful operation:

<u>Description</u>	<u>Vector Number</u>	<u>Offset</u>
Bus Error	2	\$08
Address Error	3	\$0C
Illegal Instruction	4	\$10
TRAP #15	47	\$BC
Software Abort	31	\$7C

Finally, a semaphore located at the end of the debug monitor's RAM is examined to insure that RAM has not been overwritten. If all of these conditions have been met, a warm start is initiated. A message showing the onboard RAM size and the location of the first contiguous section of offboard RAM is displayed. Note that the vectors were not initialized; any user vectors still remain, (assuming they were not the five listed above). If the vectors and RAM listed above were not correct, all vectors are reinitialized through the cold start execution path. In an effort to minimize the impact on the VMEbus by unsolicited bus errors, the debug monitor asks if RAM is to be sized and initialized to all zeros. If "N" is entered, there are no accesses to offboard resources that could generate bus errors. If "Y" or a carriage return is entered, the first contiguous section of offboard RAM is sized and initialized. The location of both onboard and the offboard RAM that was initialized is also displayed in the cold start message.

**NOTE**

If the user response is "N" to sizing and initialization, offboard memory is not accessed at this time. However, when any disk access is made (IOP, BD, etc.), offboard memory is sized and initialized before continuing.

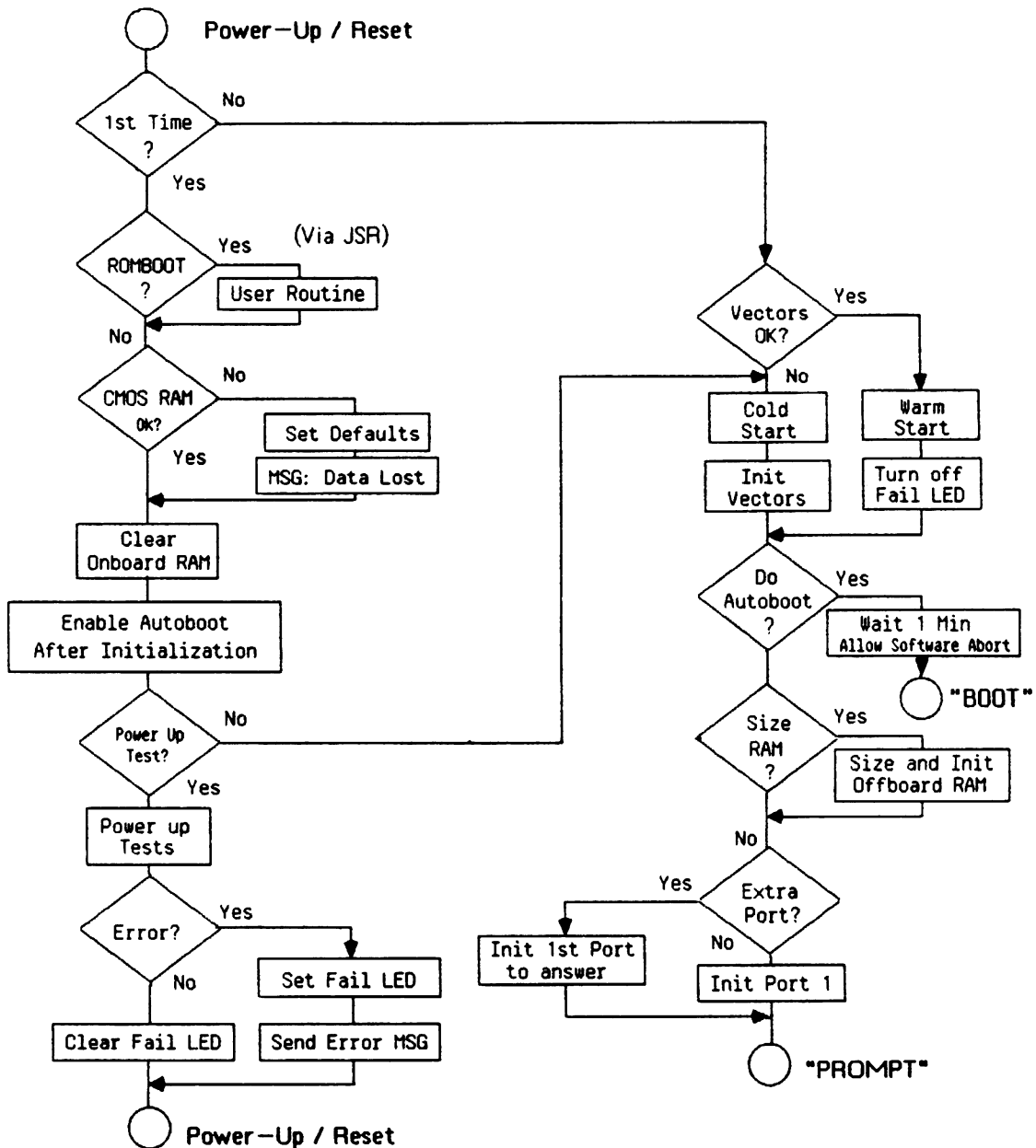


FIGURE 2-1. Power-Up/Reset

### 2.2.2 Software ABORT

When this momentary-action switch is pressed, the MVME117 enters the firmware debug monitor and displays the contents of the pseudo or target registers. The firmware monitor prompts the user, waiting for a firmware monitor debug command. After memory or registers have been examined and/or modified as required, the GO (G) command can be entered, followed by a carriage return, to continue operation of the target program that was interrupted when the software ABORT switch was pressed.

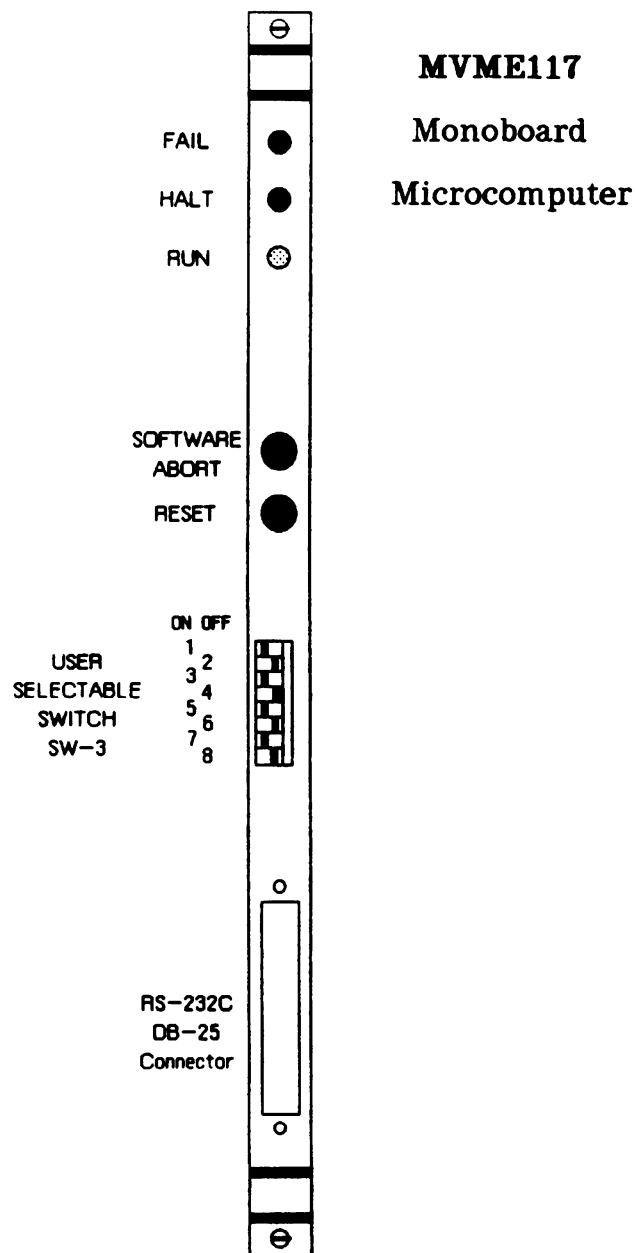


Figure 2-2. Switch Location

## 2.3 POWER-UP SEQUENCE

On Power-Up/Reset, an RS-232C terminal is assumed to be attached to the MVME117 local serial port #1. Refer to Chapter 2 of the MVME117 hardware user's manual for terminal connection and configuration.

When power is applied to the MVME117, bit 4 at location \$F44009 is set to 0 indicating that power was just applied. This bit is tested within the "Reset" logic path to see if the Power-Up Test or Autoboot facility needs to be examined. These two functions are controlled by two bytes within CMOS RAM disabled/enabled by [NO]PT and [NO]AB commands, respectively. Location \$F4400F is read, thereby setting the power-up indicator to 1 thus preventing any future Power-Up or Autoboot execution.

The first message seen by the user from the MVME117 at power-up is one of the following, depending upon the conditions set up previously by the user and the physical condition of the MVME117.

### 2.3.1 Power-Up Test Enabled

If the power-up test is enabled and none of the four conditions tested fail, there is no message displayed from the Power-Up code. Control is given to the debug monitor through the RESET vector. If a failure occurs, one of the four following messages is displayed:

"MPU TEST FAILED", "PROM CHECKSUM TEST FAILED", "MEMORY TEST FAILED", or  
"BUS Error encountered during test VME117 POWER-UP TEST FAILED".

For more information about the Power-Up test, refer to paragraph 2.6.2.

### 2.3.2 ROMBOOT Code Installed

If ROMBOOT code is installed in ROM within the memory map checked, the following message is displayed:

"Booting from ROM: xxxx"      Where xxxx are the first four bytes in the  
ASCII name of the routine given control.

For more information on the ROMBOOT feature and the requirements for building a set of ROM's, refer to paragraph 2.6.

### 2.3.3 Autoboot Enabled

If Autoboot is enabled and the drive and controller numbers provided in CMOS RAM are valid, the following message is displayed upon the system console:

"Autoboot Sequence Started... Software Abort to Interrupt"

Following this message there is about a one minute delay while the debug firmware waits for the controllers and drives to come up to speed. Then the program pointed to within the volume ID of the disk specified is loaded into RAM and control passed to it. Refer to the AB and CMOS commands in Chapter 4.

### 2.3.4 Normal Processing at Power-Up

If control is passed to the RESET routine at power-up, one of the two following messages is displayed.

- a. If the user enters N to the prompt or there is in fact no additional RAM in the VMEbus memory map, the following is displayed:

Size and Initialize RAM (Y/N) ? N

COLD Start

End Onboard RAM = \$07FFFF  
No Offboard RAM

MVME117 1.x >

- b. If the user enters Y or just presses carriage return and there is \$40000 bytes of additional memory located at \$200000, the display is:

Size and Initialize RAM (Y/N) ? Y

COLD Start

End Onboard RAM = \$07FFFF  
Start Offboard RAM = \$200000  
End Offboard RAM = \$23FFFF

MVME117 1.x >

#### NOTE

If SYSFAIL\* is activated (asserted), a warning will be displayed just prior to the display of the prompt, for both cold and warm start.

Warning: SYSFAIL\* Asserted

MVME117 1.x >

### 2.3.5 First Time Power-Up

The debug monitor validates the contents of CMOS RAM required for debug operation (refer to paragraph 1.3.1) at power-up and, if found to be bad, will set initial default values in the CMOS RAM and send the following messages:

Data lost from Battery Backed up RAM

POWER-UP Test ENABLED

AUTOBOOT from Drive 0 Controller 0 DISABLED

## 2.4 TERMINAL CONTROL CHARACTERS

Several keys are used as command line edit and control functions. The user should be familiar with these functions before using the firmware monitor. Note that the control key is indicated by CTRL. The functions include:

- a. DEL key or CTRL-H -- deletes the last character entered on the terminal, and then backspaces.
- b. CTRL-X -- cancels the entire line.
- c. CTRL-D -- redisplay the entire line.
- d. (CR) (carriage return) -- processes the current/previous command line.
- e. CTRL-S -- suspends system output to the terminal. To resume output to the terminal, any character can be entered.
- f. BREAK -- aborts commands that do any console I/O and returns to the input routine.
- g. \* (asterisk) -- Sends any text on this line to Port 2 (transparent mode for one line).

For characters requiring the control key (CTRL), the CTRL key is pressed and held down, and then the other key (H, X, D, or S) is pressed.

## 2.5 COMMAND PROCESSING

After firmware monitor initialization, the computer waits for command line input from the console terminal. A standard input routine controls the system while the user types a line of input. Command processing begins only after the line has been entered, followed by a carriage return. When a proper command is entered, the operation continues in one of two basic modes. If the command causes execution of a user program, the firmware-monitor firmware may or may not be reentered, depending on the discretion of the user. For the alternate case, the command is executed under control of the firmware monitor. During command execution, additional user input may be required, depending on the command function.

Figure 2-3 illustrates MVME117 command processing.

### NOTE

If a command causes the system to access an unused address (i.e., no memory or peripheral devices are located at that address), a bus error trap will occur. Unless default vectors have been overwritten, the terminal displays an error trap message and the contents of all MC68010 registers. Control is then returned to the firmware monitor. A bus error trap also occurs if the system attempts to write to onboard ROM.

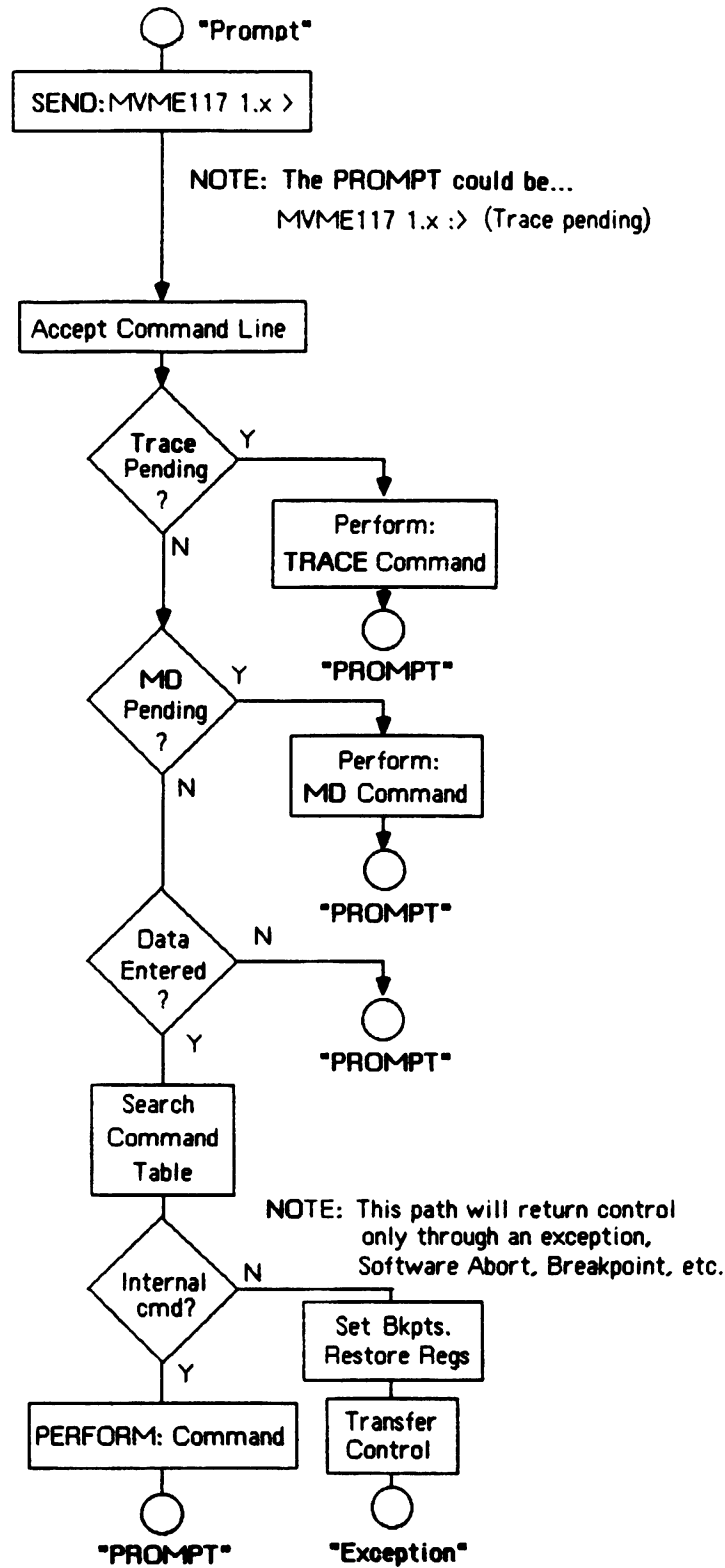


FIGURE 2-3. MVME117 Command Processing

## 2.6 ROMBOOT FACILITY

When the MVME117 completes its preliminary initialization, the power-up test location in the CMOS RAM is checked to determine whether the PWRT self-test should be executed. If not, the firmware monitor receives control immediately; if so, it receives control after execution of the self-test. After control is passed to the firmware monitor, a routine in ROM can be executed (if the ROM meets the format requirements). This feature, which provides the ability to transfer control to an external ROM routine at power-up, is named ROMBOOT. One use of ROMBOOT might be the resetting of SYSFAIL\* for an unintelligent controller module.

A module requiring the use of ROMBOOT linkage must meet the following three requirements:

- a. The routine must be located in the MVME117 memory map between addresses \$180000 to \$F40000. (This includes the two "empty" EPROM sockets.)
- b. The ASCII string "BOOT", followed by some linkage convention information, must be located on an 8K boundary within the memory range.
- c. The routine must pass a checksum test applied from the first to the last byte of the module.

To prepare a module for ROMBOOT, the CS command must be used. When the module is ready it can be loaded into RAM, and the checksum generated, installed, and verified with the CS command. (Refer to the Checksum command description and examples.)

The format of the beginning of the routine is as follows:

<u>MODULE OFFSET</u>	<u>LENGTH</u>	<u>CONTENTS</u>	<u>DESCRIPTION</u>
\$00	4	BOOT	ASCII string indicating possible routine; checksum must be zero, too.
\$04	4	Entry address	Longword offset from 8K boundary.
\$08	4	Routine Length	Longword, includes length from "BOOT" to and including checksum.
\$0C	?	Routine name	ASCII string containing routine name (only four bytes displayed).

By convention within Motorola, the last three bytes of ROM contain the firmware version number, checksum, and socket number. In this environment, the length would contain the ASCII string "BOOT" (that was on the 8K boundary), through and including the socket number; however, the user wishing to make use of ROMBOOT does not have to fill a complete ROM. Any partial amount is acceptable, as long as the length reflects where the checksum is correct.

ROMBOOT searches for possible routines starting at the high limit of memory first and checks for the "BOOT" indicator. Three events are of interest for any location being tested.

- a. If there is no memory at that location, a bus error is generated. The ROMBOOT routine is required to move on to the next 8K boundary.
- b. If memory is present, but the first four bytes do not contain BOOT, the ROMBOOT routine is required to move on to the next 8K boundary.
- c. If the ASCII string "BOOT" is located on an 8K boundary, it is still undetermined whether the routine is meant to gain control at power-up. To verify that this is the case, the bytes starting from the 8K boundary through the end of the routine (as defined by the 8K boundary + 4-byte length at offset \$8) are run through the self-test checksum routine. If both the even and odd bytes are zero, it is established that the routine was meant to be used for ROMBOOT.

The bus error routine address is replaced at location \$8 before control is passed to the routine at the point specified within the header (8K boundary + contents of offset \$4). A JSR instruction which loads the address of the next instruction within the ROMBOOT routine on the stack allows the routine to return control with an RTS to the firmware monitor following some temporary task such as initialization.

The following example returns control to the firmware monitor after placing a test pattern in a portion of RAM. Notice the use of the CS command to calculate and verify the checksum.

#### SAMPLE ROMBOOT ROUTINE - Procedure for preparing checksum

Load this relocatable ROMBOOT routine into RAM to generate checksum. R3 is an offset register used to point to the start of routine.

MVME117 1.x > MD 0+R3 40

Display (in hex) contents of RAM containing the routine.

000000+R3	42 4F 4F 54 00 00 00 10	00 00 00 2A 54 65 73 74	BOOT.....&Test
000010+R3	41 F9 00 01 F0 00 20 3C	00 00 EF FF 11 00 51 C8	Ay..p. <..o...QH
000020+R3	FF FC 4E 75 01 01 00 00	10 08 FF FF FF FF FF FF	.. Nu.....
000030+R3	FF FF FF FF FF FF FF FF	FF FF FF FF FF FF FF FF	.....

**2**

MVME117 1.x > M 10+R3;DI  
 000010+R3 41F90001F000  
 000016+R3 203C0000EFFF  
 00001C+R3 1100  
 00001E+R3 51C8FFFC  
 000022+R3 4E75  
 000024+R3 0101

LEA.L \$0001F000,A0 ? (CR)  
 MOVE.L #61439,D0 ? (CR)  
 MOVE.B D0,-(A0) ? (CR)  
 DBF.L D0,\$02001C? (CR)  
 RTS ? (CR)  
 BTST D0,D1 ? (CR)  
 0101 is revision number of routine.

000026+R3 0000

DC.W \$0000 ? (CR)  
 0000 is value to be replaced by  
 checksum.

000028+R3 1008

DC.W \$1008 ? (CR)  
 1008 are socket ID's U16 and U108.

00002A+R3 FFFF  
 00002C+R3 FFFF

DC.W \$FFFF ? (CR)  
 DC.W \$FFFF ? (CR)  
 NOTE: The socket ID's are the last  
 two bytes of the routine.

00002E+R3 FFFF  
 000030+R3 FFFF

DC.W \$FFFF ? (CR)  
 DC.W \$FFFF ? \_

MVME117 1.x > CS 0+R3 2A+R3

Calculate checksum from byte 0 to  
 byte 2A (end of routine +1). Note  
 that data located where checksum is to  
 be placed must be \$0000 at the time the  
 CS command is executed to produce  
 correct checksum bytes.

Physical Address=00020000 0002002A  
 (Even Odd)=4B34

MVME117 1.x > M 26+R3;W  
 000026+R3 0000 ? 4B34.

Enter calculated checksums with MM  
 command.

MVME117 1.x > CS  
 Physical Address=00020000 0002002A  
 (Even Odd)=0000

Issue CS command to verify zeros  
 are produced. Notice the operands  
 from the previous CS command are  
 retained for verification.

MVME117 1.x > MD 0+R3 40

Display memory one more time with  
 checksum in place.

000000+R3 42 4F 4F 54 00 00 00 10 00 00 00 2A 54 65 73 74 BOOT.....&Test  
 000010+R3 41 F9 00 01 F0 00 20 3C 00 00 EF FF 11 00 51 C8 Ay..p. <..o...QH  
 000020+R3 FF FC 4E 75 01 01 4B 34 10 08 FF FF FF FF FF FF .|Nu..K4.....  
 000030+R3 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....

MVME117 1.x >

### 2.6.1 Autoboot Support

The Autoboot (AB) command has been implemented to provide a mechanism for the user to select whether or not the MVME117 is to boot at power-up. In addition to selecting whether the feature is active or not, the user can specify the drive and controller that is to be used. Note that the drive device and controller specified with the AB command become the defaults if a Boot (BO) command is issued without specifying device or controller.

This information is kept in the CMOS RAM, which is backed up by battery to maintain the state of memory even though power has been removed. The current status of the Autoboot feature can be displayed with the use of the Help (HE) command.

At power-up, the contents of CMOS RAM are checked for valid combinations of the AB switch (Y/N) and valid drive and controller. The user is notified if data was lost from CMOS RAM. In addition, default values are inserted for future power-up cycles.

For a more detailed discussion of Autoboot and its implementation within the MVME117 Firmware Debug Monitor, refer to the AB command in Chapter 4.

### 2.6.2 Power-Up Test

A Power-Up Test (PT) command has been implemented to provide a mechanism for the user to select whether or not the MVME117 is to execute the power-up test during the power-up sequence.

This test verifies the functionality of the onboard CPU resources necessary to bring up the firmware monitor. Refer to paragraph 4.2.31 for test initiation. If the test passes, the firmware monitor comes up normally, with the FAIL LED off. If the test fails, the test is aborted when the first fault is encountered and the FAIL LED remains on. A message is output to Serial Port 1 if possible, and control is passed to the firmware monitor (see Figure 2-1). The message indicates the test module which failed.

The power-up test consists of the following modules:

a. RAM Test of Temporary Stack Area

Each longword in temporary stack, \$900 - \$AFF, is written with \$55AA55AA and verified, then written with \$AA55AA55 and verified. Parity checking is disabled during this test.

b. MC68010 MPU Test

Registers, arithmetic logic unit, instructions and exception processing are tested using the MC68010 MPU.

## c. ROM Test

A sixteen-bit checksum is calculated by the EXCLUSIVE OR of all debug ROM data.

## d. Test of Vector/firmware monitor RAM \$000 - \$8FF

The walking bit (0 through a field of 1's and 1 through a field of 0's), and pattern (\$00000000 then \$FFFFFFFF) tests are performed. Parity checking is disabled during these tests.

## e. Test of RAM \$900 - \$1100

The stack is set to an address below \$900. The walking bit (0 through a field of 1's and 1 through a field of 0's), and pattern (\$00000000 then \$FFFFFFFF) tests are performed with parity disabled.

**CHAPTER 3****COMMAND LINE FORMAT****3.1 INTRODUCTION**

Commands are entered in buffer-organized fashion. A standard input routine controls the system while the user types a line of input. Processing begins only after the carriage return has been entered.

Many primitive commands can be altered by the options field. This provides the user several extensions to the primitive commands.

Several commands are set and reset pairs; i.e., rather than having two primitive commands, the form NO is added as the first two characters of the command. For example, the set breakpoint command is BR, and the reset breakpoint command is NOBR.

The command line parser automatically converts all lowercase input into uppercase. The only exceptions are the ASCII strings within apostrophes and data entered while in transparent mode or within the LO and VE commands. These cases require actual lowercase to interface with operating systems such as SYSTEM V/68, for example, or to enter or search for lowercase strings within memory, i.e., Memory Set (MS) or Block Search (BS).

Command line formats are presented in a modified Backus-Naur Form (BNF). Certain symbols in the syntax may be used, where noted, in the real I/O. Others are metasymbols, which are used for definition only and are not entered by the user. These metasymbols and their meanings are as follows:

- < > Angular brackets enclose a word referred to as a syntactic variable, that is replaced in a command line by one of a class of items it represents.
- | This symbol indicates that a choice is to be made. One of several items, separated by this symbol, should be selected.
- [ ] Square brackets enclose an item that is optional. The enclosed item may occur zero or one time.
- [ ]... Square brackets followed by periods enclose an item that is optional/repetitive. The item may appear zero or more times.

In the examples given in the following paragraphs, operator entries are shown underscored for clarity only -- i.e., the underscore is not to be typed. Operator entries are followed by a carriage return unless otherwise specified. The carriage return is not shown in examples except where it is the only entry, in which case it is shown as (CR).

### 3.2 FIRMWARE MONITOR COMMAND LINE FORMAT

The format of the firmware monitor command line is:

```
MVME117 1.x > [NO]<command>[<port number>] [<parameters>] [;<options>]
```

where:

MVME117 1.x >	is the basic firmware monitor prompt. For prompt variations, refer to the appropriate command descriptions.
NO	is the negative form (opposite) of primitive command.
<command>	is the primitive command.
<port number>	specifies the applicable port number.
<parameters>	can be of the form <expression> or <address> and are usually separated by spaces.
<options>	specifies applicable options; multiple options may be selected.

The basic command form consists of the primitive command field and the parameters field, although some primitives do not require parameters. Some primitive commands allow specification of alternate device ports. The additional command negation and options field can modify the primitive command.

If an option exists for a command, a semicolon (;) plus <options> field(s) are added to the command. Thus, several extensions can be provided to the user.

#### 3.2.1 Expression as a Parameter

An <expression> can be one or more numeric values separated by the arithmetic operators plus (+) or minus (-). Numbers are assumed hexadecimal except for those preceded by an ampersand (&), which are decimal. In the assembler, numbers are assumed decimal unless preceded by a dollar sign (\$).

#### 3.2.2 Address as a Parameter

Many commands use <address> as a parameter. The syntax accepted by the firmware monitor is the same as that accepted by the assembler, plus a memory indirect mode. Also, contained within the firmware monitor are eight offset registers designated R0 through R7. These registers are software registers only, and are provided for easier debugging of relocatable code.

### 3.2.2.1 Address Formats.

<u>FORMAT</u>	<u>EXAMPLE</u>	<u>DESCRIPTION</u>
expression	140	Absolute address (offset register zero is added).
expression+offset	130+R5	Absolute address plus offset register five (not an assembler-accepted syntax).
expression+offset	150+R7	Absolute address (offset register seven is always zero; not an assembler-accepted syntax).
(An)	(A5)	Address register indirect.
(An,Dn) (An,An)	(A6,D4)	Address register indirect with index.
expression(An)	120(A3)	Register indirect with displacement.
expression(An,Dn) expression(An,An)	110(A2,D1)	Address register indirect with index plus displacement.
[expression]	[100]	Memory indirect (not an assembler-accepted syntax).

**3.2.2.2 Offset Registers.** Eight software registers (not actually hardware configured) are used to modify addresses contained in the firmware monitor commands. The first seven registers (.R0-.R6) are used as general-purpose offsets, while .R7 (the eighth register) is always zero. The contents of the registers can be displayed by the Offset (OF) command, paragraph 4.2.28, and modified by the .<register> command, paragraph 4.2.1.

The offset registers are always reset to zero at power-up. Thus, if their contents are not changed, the registers will have no effect on the entered address.

Unless another offset is entered, each command that expects an address parameter automatically adds offset R0 to the entered address -- that is, if R0 = 2000, the following commands are the same:

BR	10	(10 + 2000)	R0 is added by default
BR	10+R0	(10 + 2000)	
BR	2010+R7	(2010 + 0)	R7 is always zero

The physical address for each of these commands is 2010.

Offset R0 is automatically added to the offset registers any time they are modified. The only exception to this is when another offset register is specifically added. Offset registers may be set to zero by adding R7 (always zero) to zero.

EXAMPLE
COMMENT

.R0	0+R7	(R0 = 0 + 0 = 0)	R0 set to zero
.R1	8	(R1 = 8 + 0 = 8)	Offset R0 is zero, R1 is set to 8
.R0	100	(R0 = 100 + 0 = 100)	Offset R0 added
.R0	200	(R0 = 200 + 100 = 300)	Offset R0 added
.R3	100+R1	(R3 = 100 + 8 = 108)	Offset R0 not added
.R0	0+R7	(R0 = 0 + 0 = 0)	R0 set to zero

**3**
**3.3 COMMAND VERIFICATION**

As an aid to the user, the firmware monitor displays for most commands its interpretation of the values entered as expression and address parameters. The results are displayed in either physical or logical format, depending upon the command entered.

EXAMPLES:

```
MVME117 1.x > .R0 1000
MVME117 1.x > .PC 0
```

Logical Format Example

```
MVME117 1.x > MD 0
000000+R0 4E 71 4E 71 4E 71 4E 71 4E 71 00 00 0F 90 00 00  NqNqNqNqNq.....
```

Physical Format Example

```
MVME117 1.x > GT 8
Physical Address=00001008
Physical Address=00001000
```

At Breakpoint

```
PC=00001008 SR=2700=.S7..... USP=00012C5C SSP=0000085E VBR=00000000 SFC=1
DFC=0
D0-7 00304E71 00001000 4E711000 00000000 00004E71 0000002C 00001008 00000000
A0-7 000004DA 00000000 00001000 0000053A 00001002 00000551 00000551 0000085E
PC=000008+R0 4E71 NOP
```

Commands entered are also checked for validity. For example, specifying an address parameter which would result in an error may cause the message INVALID ADDRESS=xxxxxxx to be displayed on the console terminal. Or, if a hex address contains a "non-hex" character (not between 0 through 9 and A through F), "ERROR" is displayed.

A table of firmware monitor error messages is provided in Appendix B.

## CHAPTER 4

### COMMAND SET

#### 4.1 INTRODUCTION

Chapter 4 describes the command line syntax and provides one or more examples for each command in the firmware monitor command set. Table 4-1 lists the firmware monitor mnemonics by type.

TABLE 4-1. Firmware Monitor Commands by Type

COMMAND MNEMONIC	DESCRIPTION
MD	Memory display/disassembly
M[M]	Memory modify/disassembly/assembly
MS	Memory set
.A0-.A7	Display/set address register
.D0-.D7	Display/set data register
.DFC	Display/set destination function code
.PC	Display/set program counter
.SFC	Display/set source function code
.SR	Display/set status register
.SS[P]	Display/set supervisor stack pointer
.US[P]	Display/set user stack pointer
.VBR	Display/set vector base register
DF	Display formatted registers
BF	Block of memory fill
BI	Block initialize
BM	Block of memory move
BS	Block of memory search
BT	Block of memory test
DC	Data conversion
VI	Vector initialize
.R0-.R6	Display/set relative offset register
.R7	Display relative offset register
OF	Display offsets

**TABLE 4-1. Firmware Monitor Commands by Type (cont'd)**

COMMAND MNEMONIC	DESCRIPTION
BR	Breakpoint set
NOBR	Remove breakpoint
G[0]	Execute program
GT	Go until breakpoint
GD	Go direct execute program
T[R]	Trace
TT	Trace to temporary breakpoint
PA	Printer attach
NOPA	Detach printer
TA	Terminal attach
CS	Calculate checksum
PF	Port format
PT	Enable power-up test
NOPT	Disable power-up test
ST	Self-test
SET	Set date/time
TIME	Display date/time
TM	Transparent mode
HE	Help
DU	Dump memory (S-records)
LO	Load (S-records)
VE	Verify (S-records)
AB	Enable Autoboot
NOAB	Disable Autoboot
BD	Boot dump
BH	Boot halt
BO	Boot load
CMOS	Display CMOS RAM variables
RESET	SCSI/SASI bus reset or SCSI controller reset
IOC	I/O command for drive/controller
IOP	I/O physical for drive/controller
IOT	I/O teach for drive/controller

## 4.2 FIRMWARE MONITOR COMMANDS

A complete description of each firmware monitor command is provided in the following paragraphs. Messages resulting from error conditions during command execution are described in Appendix B.

### 4.2.1 Display/Set Register (.<register>)

.&lt;register&gt;

.&lt;register&gt; [&lt;expression&gt;]

The .<register> commands allow the user to display or modify individual registers. Commands with a leading period and the registers displayed/alterd by these commands are:

.A0-.A7	address register
.D0-.D7	data register
.DFC	destination function code (used with MC68010 MOVES instruction)
.PC	program counter
.R0-.R6	relative offset register (software register) (refer to OF command)
.R7	relative offset register seven (software register) (always 0; cannot be set; can be read) (refer to OF command)
.SFC	source function code (used with MC68010 MOVES instruction)
.SR	status register (in the MC68010)
.SS[P]	supervisor stack pointer
.US[P]	user stack pointer
.VBR	vector base register

#### EXAMPLE

#### COMMENT

```
MVME117 1.x > .PC
.PC=00000A00
```

Display program counter.

```
MVME117 1.x > .A7 F00
```

Set address register 7 to \$F00.

```
MVME117 1.x > .R1 A00
```

Set relative offset register 1 to \$A00.

```
MVME117 1.x > OF
```

Display all relative offset registers.

```
R0-7 00000000 00000A00 00000000 00000000 00000000 00000000 00000000 00000000
```

```
MVME117 1.x > DF
```

Display all formatted CPU registers.

```
PC=00000A00 SR=2700=.S7.....USP=FFFFFFFF SSP=00000F00 VBR=00000000 SFC=2 DFC=7
D0-7 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000F00
PC=000000+R1 41F81000 LEA.L $00001000,A0
```

## 4.2.2 Autoboot Enable/Disable (AB and NOAB)

AB  
NOAB

AB [<device>,<controller>]  
NOAB

where:

<device> is a single hexadecimal digit (0-7) specifying the device or drive to be used. (Default = 0 or previous drive value.)

<controller> is a single hexadecimal digit (0-8) specifying the controller to be used. (Default = 0 or previous controller value.)

4

The function of the Autoboot (AB) command is to provide a mechanism to Enable/Disable Autoboot upon power-up. In addition, the default drive and controller to be used can be entered with the AB command.

CMOS RAM, which is battery backed-up, is used to contain the enable/disable status as well as the drive and controller to be used. The actual addresses used are provided in the following list:

\$F43FC5 = Check for AB at prompt?	ASCII 'Y' = Check for Autoboot
	ASCII 'N' = Do not check for Autoboot
\$F43FC7 = Local level (SCSI)	\$80 = Level 7 (Default starting value)
\$F43FC9 = Power-Up Switch	ASCII 'Y' = Power-Up enabled (Default)
	ASCII 'N' = Power-Up disabled
\$F43FCB = Autoboot Switch	ASCII 'Y' = Autoboot enabled
	ASCII 'N' = Autoboot disabled (Default)
\$F43FCD = Drive Number	ASCII '0' thru '7' (Default is '0')
\$F43FCF = Controller Number	ASCII '0' thru '8' (Default is '0')

### NOTES

1. Only odd addresses are provided in CMOS RAM. An \$F is displayed for each even address when Memory Display (MD) command is used.
2. When using a 2K x 8 CMOS RAM, such as the one installed at the factory, its addresses are echoed at \$F40Fxx, \$F41Fxx, \$F42Fxx, and \$F43Fxx. However, this is not true when an 8K x 8 CMOS RAM is installed. A single address space is displayed and used.

If for some reason the contents of these locations are destroyed, (due to battery failure or data being moved over these locations), the next time power is applied the message:

Data lost from Battery Backed-up RAM  
Autoboot from Drive 0 Controller 0 DISABLED

AB  
NOAB

is displayed informing the user of the event. If the current contents of these locations are desired, the Help (HE) command can be invoked to have this information presented in a formatted display. All of CMOS RAM, (used to save debug monitor data), can be displayed with the use of the CMOS command.

If Autoboot is enabled at power-up, the message "Autoboot started ..." is displayed on the system console. After approximately 60 seconds, the actual disk I/O is begun. If, however, the user wants to gain control without Autoboot, the Software ABORT or RESET keys can be pressed. In addition to allowing operator intervention, the delay allows enough time for various controllers and drives to become ready.

If VERSAdos is being "booted", and Autoboot is no longer desired, the following procedure can be used to disable Autoboot:

- a. Once the VERSAdos initialization is complete and the prompt has been displayed, press the Software ABORT key.
- b. Now the NOAB command can be entered, disabling Autoboot.
- c. The next time power is applied, Autoboot is not invoked, allowing direct entry to the MVME117 debug monitor.

#### NOTE

Not all operating systems may support Software Abort and the user may have to use the method discussed previously (during the 60-second delay) to gain access to the MVME117 debug monitor once Autoboot is enabled.

#### EXAMPLE

MVME117 1.x > AB 8,0  
Drive Number not 0 thru 7

MVME117 1.x > ab 0,9  
Controller Number not 0 thru 8

MVME117 1.x > AB 0,0  
AUTOBOOT from Drive 0 Controller 0 ENABLED

MVME117 1.x > NOAB  
AUTOBOOT from Drive 0 Controller 0 DISABLED

#### COMMENT

Command requesting Autoboot  
Enable for bad drive number.

Command requesting Autoboot  
Enable for bad controller  
number.

Command requesting Autoboot  
Enable for drive 0 controller  
0.

Command requesting Autoboot  
Disable.

AB  
NOAB

### EXAMPLE

MVME117 1.x > AB  
AUTOBOOT from Drive 0 Controller 0 ENABLED

MVME117 1.x > ab 2 1  
AUTOBOOT from Drive 2 Controller 1 ENABLED

MVME117 1.x > noab  
AUTOBOOT from Drive 2 Controller 1 DISABLED

MVME117 1.x > ab  
AUTOBOOT from Drive 2 Controller 1 ENABLED

MVME117 1.x > HE

POWER-UP test ENABLED

AUTOBOOT from Drive 2 Controller 1 ENABLED

.PC .SR .US .SS .VBR .DFC .SFC  
.D0 - .D7  
.A0 - .A7  
.R0 - .R7

CMOS	IOC	IOP	IOT	RESET	SET	TIME						
AB	NOAB	BD	BF	BH	BI	BM	BO	BR	NOBR	BS	BT	
CS	DC	DF	DU	G	GD	GO	GT	HE	LO	M	MD	
MM	MS	OF	PA	NOPA	PF	PT	NOPT	ST	T	TA	TM	
TR	TT	VE	VI									

### -PORTS-

MVME117 MVME050 MVME400 MVME410  
1=Term1 4=Term1 7=Term2 9=PRT A  
2=Term2 5=Term2 8=Term1 A=PRT B  
3=PRT 6=PRT

MVME117 1.x >

### COMMENT

Re-enable Autoboot using previous drive & controller.

Enable Autoboot using drive 2 controller 1. Note lower case and blank delimiter.

Disable Autoboot.

Re-enable Autoboot using previous drive and controller.

Display current Autoboot status, drive and controller.

-Disk Controllers-  
SCSI=0 thru 7  
RWIN=8

### 4.2.3 Boot Dump (BD)

**BD**

BD [<device>] [,<controller>]

where:

<device> is a single hexadecimal digit (0 through 7) specifying the device to be used (default = Autoboot drive).

<controller> is a single hexadecimal digit (0 through 8) specifying the controller to which the device is connected (default = Autoboot controller).

The BD command dumps the contents of memory to an allocated area on disk.

The BD command causes the volume ID (sector zero) of the device and controller specified to be read and examined. The location on the media where the contents of memory will be written is obtained from the volume ID.

The bootstrap dump sequence first reads sector zero of the specified media, verifies that the disk is not foreign by either character string "EXORMACS" or "MOTOROLA" in the last eight bytes, and reads the parameters of the media location where the data will be written. Sector zero locations are:

\$84 through \$87 = sector to start writing on  
\$88 through \$89 = number of sectors allocated

The following are examined to locate the smaller number of sectors that will be written to disk:

- a. Number of sectors previously allocated (from volume ID).
- b. The number of sectors required to contain all the contiguous memory that the firmware monitor initialized. Refer to paragraph 2.2.1, RESET.

The dump proceeds from address \$0 to the end of contiguous memory or the end of dump area, whichever comes first. Note that any gap that exists between onboard and offboard RAM reserves space within the allocated dump file on the drive specified.

#### EXAMPLE

#### COMMENT

MVME117 1.x > BD 0,3

Onboard RAM Blocks = \$0800  
GAP Blocks = \$0000  
Offboard RAM Blocks = \$0000  
Total Required = \$0800  
BD Blocks Allocated = \$0600

The first example illustrates how users can change their minds after seeing that enough disk space was not allocated at "INIT" time.

Note this configuration has no offboard RAM. \$800 blocks are required for a complete image dump, but only \$600 were allocated.

Are you sure, (Y,N) ? N  
MVME117 1.x >

Do not proceed.  
Note that any character other than "Y" will terminate BD and return the prompt.

BD

**EXAMPLE**

```
MVME117 1.x > bd 0,3

Onboard RAM Blocks = $0800
      GAP Blocks = $1800
Offboard RAM Blocks = $0400
      Total Required = $2400
BD Blocks Allocated = $2400
```

```
Are you sure, (Y,N) ? y
```

```
Dump Ok
MVME117 1.x >
```

**COMMENT**

The second example illustrates a normal BD command with enough space allocated to accommodate not only the onboard RAM but also the first offboard section as well.

"y" entered -- note the use of lowercase is supported.

```
MVME117 1.x > ab 0,3
AUTOBOOT from Drive 0 Controller 3 ENABLED
```

```
MVME117 1.x > noab
AUTOBOOT from Drive 0 Controller 3 DISABLED
```

```
MVME117 1.x > bd

Onboard RAM Blocks = $0800
      GAP Blocks = $0000
Offboard RAM Blocks = $0000
      Total Required = $0800
BD Blocks Allocated = $0600
```

```
Are you sure, (Y,N) ? y
```

```
PARTIAL Dump completed
MVME117 1.x >
```

The third example uses the default Autoboot parameters: set drive 0 and controller 3.

Now that the drive and controller are set, disable Autoboot.

The Autoboot drive and controller are now available and are used because no drive and controller were entered.

Note that in this example there is no offboard RAM and there were not enough blocks allocated for an entire image dump. The user has chosen to go ahead and dump all but the last \$200 blocks.

The BD command indicates that only a partial dump was taken.

#### 4.2.4 Block Fill (BF)

**BF**

BF <address1> <address2> <pattern>

The BF command fills a specified block of memory with a specified binary pattern of word size. A word boundary (even address) must be given for the starting <address1> and ending <address2> of the block. The pattern word may be expressed in hexadecimal (default), decimal, octal, or binary format. Refer to the DC command for symbols used to denote numeric type. If a pattern of less than word size is entered, the data is right-justified and leading zeros are inserted by the firmware monitor.

##### EXAMPLE

##### COMMENT

```

MVME117 1.x > MD 1000 20          Show current RAM contents.
001000    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
001010    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

MVME117 1.x > BF 1000 100E 4E71    Block fill with $4E71 (NOP).
Physical Address=00001000 0000100E

MVME117 1.x > MD 1000 20          Show same locations after block fill.
001000    4E 71 4E 71 4E 71 4E 71 4E 71 4E 71 4E 71 NqNqNqNqNqNqNq
001010    00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

MVME117 1.x > MD 1000 4;DI        Display same area using disassembler
001000    4E71                      NOP      (first 4 instructions only).
001002    4E71                      NOP
001004    4E71                      NOP
001006    4E71                      NOP
MVME117 1.x >

```

#### 4.2.5 Boot Halt (BH)

**BH**

BH [<device>][,<controller>]

where:

<device> is a single hexadecimal digit (0-7) specifying the device to be used (default = Autoboot drive).

<controller> is a single hexadecimal digit (0-8) specifying the controller to which the device is connected (default = Autoboot controller).

The BH command causes data from disk to be loaded into memory and program control to be given to the debug monitor. If device and/or controller are not specified, the device and controller currently assigned as Autoboot values are used. Refer to the Help (HE) command.

This command works the same as B0, except that control is transferred back to the debug monitor.

See also: B0

#### EXAMPLE

#### COMMENT

MVME117 1.x > AB 2,0 Establish default device and controller.

AUTOBOOT from Drive 2 Controller 0 ENABLED

MVME117 1.x > BH Boot and Halt command without operands.  
(Defaults are drive 2 and controller 0 from the preceding AB command.)

Booting from: FLOP  
Into RAM at: \$010000

PC=00010894 SR=2700=.S7.....USP=00000000 SSP=00050000 VBR=00000000 SFC=0 DFC=0  
D0-7 00000032 00000030 00000000 0000000A 49504C03 00010000 00000000 00000059  
A0-7 00000000 00010894 0000147E 00F00E86 00F006DC 000012A0 000012A0 00050000  
PC=010894 46FC2700 MOVE.W #9984,SR

MVME117 1.x > BH 3 Boot and Halt from drive 3. (Default controller is still 0 from above.)

Booting from: TS15  
Into RAM at: \$002000

PC=00002008 SR=2700=.S7.....USP=00000000 SSP=00002500 VBR=00000000 SFC=0 DFC=0  
D0-7 00000003 00000030 00000000 0000000A 49504C03 00002000 00000000 00000059  
A0-7 00000000 00002008 0000147E 00F00E86 00F006DC 000012A2 000012A2 00002500  
PC=002008 4BFA0068 LEA.L \$00002072(PC),A5

MVME117 1.x >

#### NOTE

To use the BH command, a valid stack value must be in locations \$0-\$3 of the file being loaded.

#### 4.2.6 Block Initialize (BI)

BI

BI &lt;address1&gt; &lt;address2&gt;

The BI command initializes word parity in a specified block of memory consisting of <address1> through <address2>. No data in any word is changed if parity in the word is correct. If parity in a word is incorrect, the characters "m?" (\$6D3F) are written in that word to force correct parity. If the parity cannot be set in one or more words, the message BUS ERROR TRAP is displayed on the console. The BT (Block Test) command may be used to isolate the failure(s).

#### NOTE

Both addresses must be on word boundaries  
(even addresses).

See also: BT

#### EXAMPLE

MVME117 1.x > BI 44000 4FFFE  
Physical Address=00044000 0004FFFE

MVME117 1.x >

### 4.2.7 Block Move (BM)

**BM**

BM <address1> <address2> <address3>

where:

<address1> is the starting address of the source memory block.

<address2> is the ending address of the source memory block.

<address3> is the starting address of the destination memory block.

The BM command is used to move (duplicate) blocks of memory from one area to another.

#### EXAMPLE

MVME117 1.x > MD 1B00 4;DI

001B00 1018

001B02 0C000000

001B06 67F8

001B08 4E75

MOVE.B (A0)+,D0

CMP.B #0,D0

BEQ.S \$001B00

RTS

MVME117 1.x > MD 1A00 3;DI

001A00 FFFF

001A02 0000FFFF

001A06 0020FFFF

DC.W \$FFFF

OR.B #-1,D0

OR.B #-1,-(A0)

MVME117 1.x > BM 1B00 1B09 1A00

Physical Address=00001B00 00001B09

Physical Address=00001A00

MVME117 1.x > MD 1A00 4;DI

001A00 1018

001A02 0C000000

001A06 67F8

001A08 4E75

MOVE.B (A0)+,D0

CMP.B #0,D0

BEQ.S \$001A00

RTS

MVME117 1.x >

#### 4.2.8 Boot Load (B0)

**B0**

B0 [<device>][,<controller>][,<string>]

where:

- <device> is a single hexadecimal digit (0-7) specifying the device to be used (default = Autoboot device).
- <controller> is a single hexadecimal digit (0-8) specifying the controller to which the device is connected (default = Autoboot controller).
- <string> is an optional ASCII character string that is passed to the program being loaded from the device and controller.

The function of the B0 command is to access a program on disk, transfer it into memory space, and give control to that program. Where to find the program and where in memory to store the program is contained in sector 0 of the disk corresponding to the specified device and controller. If the device and controller are not specified, the default value is used for each.

The following sequence occurs when the B0 command is executed:

- a. Starting at sector 0 (the volume ID), 256 bytes are read and transferred into the firmware monitor workspace RAM.
- b. If the volume ID (locations \$0 through \$3) is not null, these four ASCII bytes will be displayed as follows:

```
Booting from:  SYS
Into RAM at:  $010000
```

where SYS is the volume ID. If null, the display is suppressed.  
 \$01000 is the load address extracted from location  
 \$1E - \$21 of the volume ID.

- c. Motorola ID locations \$F8-\$FF are read to ensure that they contain either "EXORMACS" or "MOTOROLA". If not, the error message "Foreign Media - Command aborted" appears, and the prompt is returned.
- d. The location of the program to be loaded and its destination in memory are identified by examining the first sector at the following locations.

<u>LOCATIONS</u>	<u>CONTENTS</u>
\$14-\$17	First 256-byte sector to transfer
\$18-\$19	Number of sectors to transfer
\$1E-\$21	Address of first destination byte (first memory address)

4

- | <u>LOCATIONS</u> | <u>CONTENTS</u>  |
|------------------|--|
| \$90-\$93        | Sector address of the media configuration parameters (normally sector 1) |
| \$94             | Length of the configuration area (normally one 256-byte sector)          |

- f. The program is read and transferred to its memory destination.
- g. The status register is updated to reflect supervisor mode and interrupt level 7.
- h. The stack pointer is loaded from locations \$0-\$3 relative to the destination memory.
- i. The program counter is loaded from locations \$4-\$7 relative to the destination memory.

```

D0...Drive number
D1...Controller number
D2...Winchester controller byte 5
D3...ME4U coded key for IPL
D4...Flags for IPL; IPLx where x = bits 7654 3210
    Firmware support for TRAP #15                x
    Firmware support IPL Disk I/O                x
    Unused (Reserved)                        xxxx xx
A0...Address of controller hardware
A1...Address of program just loaded
A2...Address of configuration work area just used
A3...Address of DISKR for IPL and others (disk read subroutine)
A4...Address of the firmware monitor entry point
A5...Start of string (beyond drive and controller)
A6...End of string (where next byte would go)
A7...Stack of program just loaded
SR...2700 (supervisor mode at level 7)

```

**B0**

These registers can be used by IPL to load the file identified by the string field. If a string field is specified on the B0 command line, registers A5 and A6 point to the first and last plus one characters of the string. If no string is specified, register A5 = A6.

For information regarding use of the string within VERSAdos, refer to the discussion of the bootload file, IPL.SY, in the M68000 Family VERSAdos System Facilities Reference Manual.

The devices and controllers currently supported by the firmware monitor are assigned as follows:

<u>DEVICE #</u>	<u>CONTROLLER</u>	<u>DESCRIPTION</u>
0-7	0-7	SCSI controllers and drives
0	8	Winchester hard disk
1	8	Winchester hard disk
2	8	5-1/4 inch Winchester floppy disk
3	8	5-1/4 inch Winchester floppy disk

NOTE: Controller 8 is a Motorola RWIN controller. Refer to paragraph 1.3.3.2.

EXAMPLE
COMMENT

<p>MVME117 1.x &gt; <u>NOAB</u></p> <p>AUTOBOOT from Drive 3 Controller 0 DISABLED</p> <p>MVME117 1.x &gt; <u>B0</u></p> <p>Booting from: TS15 Into RAM at: \$002000</p> <p>MVME117 1.x &gt; <u>B0 2</u></p> <p>Booting from: SYS Into RAM at: \$010000 Boot in progress ... Boot complete</p> <p>MVME117 1.x &gt; <u>B0 0,8,OS.SY</u></p> <p>Booting from: SYS Into RAM at: \$010000 Boot in progress ... Boot complete</p>	<p>This shows the Autoboot default parameters.</p> <p>Boot using both default parameters (drive 3 and controller 0). Volume ID in drive 3. RAM location where "non-IPL" program is to be loaded.</p> <p>Boot using drive 2 and default controller 0. Volume ID in drive 2. RAM location where IPL is being loaded.</p> <p>Boot using drive 0 and controller 8, and request that IPL locate and load a program named OS.SY.</p>
--	--

#### 4.2.9 Breakpoint Set and Remove (BR and NOBR)

BR  
NOBR

```
BR (display only)
BR [<address>[;<count>]] [<address>[;<count>]]...
NOBR [<address>[<address>]...]
```

When encountered, a breakpoint causes target program execution to stop and control to be transferred to the firmware monitor. The BR command may be used without parameters to cause display of current breakpoint addresses. The BR <address> command sets one or more addresses into the breakpoint address table. This table can hold up to eight breakpoint addresses. Multiple breakpoints (up to eight) may be specified with one call of the Breakpoint command. Addresses should be on even word boundaries. The range of <count> is a 32-bit integer.

The breakpoints are inserted into the target program when execution is called via a GO or GT command. The illegal instruction \$4AFB is inserted at the addresses specified by the table. During execution of the program, a breakpoint occurs whenever this instruction is encountered. If program control is lost, control may be regained via the RESET or the ABORT switch. ABORT is preferred because use of the RESET function may leave breakpoints (\$4AFB) in the user program, whereas ABORT will recover properly (refer to Appendix A).

While executing a Trace command, the breakpoint addresses are monitored (i.e., the illegal instruction \$4AFB is not placed in memory).

After stopping at a breakpoint, execution may be continued by typing the GO command.

The NOBR command removes one or more breakpoints from the internal breakpoint table. The NOBR command without parameters eliminates all breakpoints.

##### BR COMMAND FORMAT

##### DESCRIPTION

MVME117 1.x > BR

Display all breakpoints.

MVME117 1.x > BR <address>

Set a breakpoint.

MVME117 1.x > BR <address>;<count>

Set a breakpoint with a count. Count is decremented each time the breakpoint is encountered until <count> = 0. Execution stops as soon as count is decremented to zero. Thereafter, execution will stop each time the breakpoint is reached.

BR  
NOBR

### NOBR COMMAND FORMAT

### DESCRIPTION

MVME117 1.x > NOBR

Clear all breakpoints.

MVME117 1.x > NOBR <address>

Clear a specific breakpoint.

See also: GT, PF (Options @ xxxx), and TT.

### EXAMPLE

MVME117 1.x > .R4 4000

MVME117 1.x > BR 1010 2000;5 2040 4000

Breakpoints

```
001010    001010
002000    002000;5
002040    002040
000000+R4 004000
```

MVME117 1.x > NOBR 1010 2040

Breakpoints

```
002000    002000;5
000000+R4 004000
```

MVME117 1.x > NOBR

Breakpoints

MVME117 1.x >

#### 4.2.10 Block of Memory Search (BS)

**BS**

```
BS <address1> <address2> '<literal string>'
BS <address1> <address2> <data> [<mask>] [;<option>]
```

The BS command has two modes: literal string search and hex data search. Both modes can search memory beginning at <address1> through <address2>, looking for a match. Alternatively, a user can specify that a data search report back only locations that do not match the input data. This alternative search for a mismatch can be particularly useful when searching for suspected faulty memory. For example, a known pattern can be placed into suspect RAM locations, and a BS command with an option to search for a mismatch will display any bad RAM locations.

The literal string mode is initiated if a single quote (') follows <address2>. The ASCII literal string may contain both uppercase and lowercase letters. If a single quote does not follow <address2>, data search mode is assumed. If the optional mask is supplied with a data search, the mask is ANDed to the data found at each address. The data located in the memory is not changed. The masked data is then examined for a match. (The default mask is all 1's.)

Available options for a data search enable a user to specify the data format and whether to search for a match or a mismatch. The options to specify data format are the letters B, W, and L. To specify a mismatch, a minus sign is placed before or after the data format indicator. If there is no minus sign in the options field, a matching search is assumed.

```
;B      Data format is a byte; search for a match.
;-B     Data format is a byte; search for a mismatch.
;W      Data format is a word; search for a match.
;-W     Data format is a word; search for a mismatch.
;L      Data format is a longword; search for a match.
;-L     Data format is a longword; search for a mismatch.
```

The default value for a data search is ;B.

When a search is completed, each address containing data that meets the specified requirements is displayed on the terminal, along with the data located at that address.

To illustrate the searching command, the following examples are provided:

##### EXAMPLE

##### COMMENT

```
MVME117 1.x > MD 10000 40          Show memory to be searched.
010000  A5 5A A5 5A A5 5A A5 5A  A5 5A A5 5A A5 5A A5 5A  %Z%Z%Z%Z%Z%Z%Z%Z
010010  41 61 20 41 42 61 62 20  41 42 43 61 62 63 20 20  Aa ABab ABCabc
010020  A5 5A A5 5A A5 5A A5 5A  A5 5A A5 5A A5 5A A5 5A  %Z%Z%Z%Z%Z%Z%Z%Z
010030  A5 5A A5 5A A5 5A A5 52  A5 52 A5 52 A5 5A A5 5A  %Z%Z%Z%R%R%R%Z%Z
```

BS

EXAMPLE

MVME117 1.x > BS 10000 10040 'ab'  
Physical Address=00010000 00010040  
010015 'ab'  
01001B 'ab'

MVME117 1.x > BS 10000 10040 43 DF;B  
Physical Address=00010000 00010040  
01001A 43  
01001D 63

MVME117 1.x > BS 10020 10040 A55A;-W  
Physical Address=00010020 00010040  
010036 A552  
010038 A552  
01003A A552

COMMENT

Successful search for literal string 'ab'.

Successful data search using a mask allowing both lowercase and uppercase ASCII C.

Search for any words NOT matching the test pattern.

4

#### 4.2.11 Block of Memory Test (BT)

BT

BT &lt;address1&gt; &lt;address2&gt;

The BT command provides a destructive test of a block of memory. A word boundary (even address) must be given for the starting <address1> and ending <address2> of the block. If the test runs to completion without detecting an error, all memory tested will have been set to zeros.

Execution of this command may take several seconds for large blocks of memory.

When a problem is found in a memory location, the address, the data stored, and the data read are displayed. Control is then returned to the firmware monitor.

See also: BI

##### EXAMPLE

MVME117 1.x > BT 44000 47FFE  
Physical Address=00044000 00047FFE

##### COMMENT

Successful memory test; no errors found.

MVME117 1.x > BT 44000 4FFFE  
Physical Address=00044000 0004FFFE  
FAILED AT 0480FE WROTE=FFFF READ=0000

Unsuccessful memory test; error data is listed.

MVME117 1.x >

## 4.2.12 Display CMOS RAM Variables (CMOS)

**CMOS**

### CMOS

The CMOS command provides an easy way to display the portion of CMOS RAM reserved for the debug monitor variables. By entering this command, \$F00 bytes are displayed upon the screen by the MD command. (Note that if a carriage return is entered following the CMOS command, the next \$F00 bytes are displayed just as if MD had been entered by the user.) Data is saved in odd RAM locations only, with \$FF displayed for each even RAM location.

The specific variables contained within CMOS RAM are as follows:

\$F43F21	SCSI controller type for SCSI ID Level 00	(Default of 2)
\$F43F23	SCSI controller type for SCSI ID Level 01	(Default of 8)
\$F43F25	SCSI controller type for SCSI ID Level 02	(Default of 2)
\$F43F27	SCSI controller type for SCSI ID Level 03	(Default of 8)
\$F43F29	SCSI controller type for SCSI ID Level 04	(Default of 2)
\$F43F2B	SCSI controller type for SCSI ID Level 05	(Default of 8)
\$F43F2D	SCSI controller type for SCSI ID Level 06	(Default of 2)
\$F43F2F	SCSI controller type for SCSI ID Level 07	(Default of Null)
\$F43F41	First of two Attach packet save areas	(Compressed)
\$F43F81	Second of two Attach packet save areas	(Compressed)
\$F43FC5	Check for Autoboot at prompt Y or N	
\$F43FC7	Local ID level code for the MVME117	(Default of \$80)
\$F43FC9	Power-Up indicator Y or N	(Default of Y)
\$F43FCB	Autoboot indicator Y or N	(Default of N)
\$F43FCD	Autoboot drive number 0-7	(Default of 0)
\$F43FCF	Autoboot SCSI controller level number 0-8	(Default of 0)
\$F43FD1-\$F43FDF	Date when last SET command was entered	
\$F43FE1-\$F43FEF	Time when last SET command was entered	

NOTES on Autoboot (AB) command, paragraph 4.2.2, have additional discussion of address differences in 2K x 8 CMOS RAM and in 8K x 8 CMOS RAM.

### EXAMPLE

```

MVME117 1.x > CMOS
F43F20 FF 32 FF 38 FF 32 FF 38 FF 32 FF 38 FF 32 FF 26 .2.8.2.8.2.8.2.&
F43F30 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 .....
F43F40 FF FF FF FF FF FF FF FF FF 00 FF 00 FF 00 FF 00 .....
F43F50 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 .....
F43F60 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 .....
F43F70 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 .....
F43F80 FF FF FF FF FF FF FF FF FF 00 FF 00 FF 00 FF 00 .....
F43F90 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 .....
F43FA0 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 .....
F43FB0 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 .....
F43FC0 FF 00 FF 00 FF 4E FF 80 FF 59 FF 4E FF 30 FF 30 .....N...Y.N.0.0
F43FD0 FF 31 FF 30 FF 2F FF 31 FF 30 FF 2F FF 38 FF 35 .1.0./1.0./8.5
F43FE0 FF 30 FF 39 FF 3A FF 35 FF 39 FF 3A FF 35 FF 35 .0.9.:5.9.:5.5
MVME117 1.x >

```

#### 4.2.13 Calculate Checksum (CS)

CS

CS [&lt;address1&gt;] [&lt;address2&gt;]

The Checksum command provides access to the same checksum routine used by the power-up self-test firmware. This routine is used in two ways within the firmware monitor.

- a. At power-up, if the Power-Up Test is enabled, the Power-Up Test is executed. One of the items verified is the checksum contained in the firmware monitor ROM. If for any reason the contents of ROM were to change from the factory version, the checksum test is designed to detect the change and inform the user of the failure.
- b. Following a valid power-up test, MVME117 1.x examines the VME address space for code that needs to be executed. This feature (ROMBOOT) makes use of the checksum routine to verify that a routine in memory is really there to be executed at power-up. For more information refer to paragraph 2.6, which describes the format of the routine to be executed and the interface provided upon entry.

This command is provided as an aid in preparing routines for the ROMBOOT feature. Since ROMBOOT does checksum validation as part of its screening process, the user needs access to the same routine in the preparation of EPROM/ROM routines.

The [<address>] parameters can be provided in two forms:

- a. An absolute address (24-bit maximum).
- b. An expression using a displacement + relative offset register.

Any previous addresses are saved as default addresses for CS commands invoked later. This is convenient since users typically enter the address range to calculate the checksum and then enter the results into memory (into bytes that were \$0000 while the checksum was calculated). When the CS command is used to verify the content and location of the new checksum, the operands need not be entered since the command retains the addresses used to calculate the previous checksum. The even and odd byte result should be 0000, verifying that the checksum bytes were calculated correctly and placed in the proper locations.

The default operands at power-up are the starting/ending addresses of the MVME117 1.x firmware. The results for even and odd bytes should be 0000.

CS

The algorithm used to calculate the checksum is as follows:

- a. \$FF is placed in each of two bytes within a register. These bytes represent the even and odd bytes as the checksum is calculated.
- b. Starting with the first address, the even and odd bytes are extracted from memory and XORed with the bytes in the register.
- c. This process is repeated, word by word, until the ending address is reached. Note that the last word addressed is NOT included in the checksum. This technique allows use of even ending addresses (\$D40000 as opposed to \$D3FFFE).

CS

EXAMPLE

MVME117 1.x > CS
COMMENT

CS command entered without operands right after power-up (addresses are firmware monitor limits by default).

Physical Address=00F00000 00F08000  
(Even Odd)=0000

MVME117 1.x > MD 20000 3F

Display routine requiring a checksum. Start at \$20000; last byte is at \$20029. Checksum will be placed in bytes at \$20026 and \$20027, so they are zero while calculating the checksum.

```
020000  42 4F 4F 54 00 00 00 14  00 00 00 A6 54 65 73 74  BOOT.....&Test
020010  41 F9 00 01 F0 00 20 3C  00 00 EF FF 11 00 51 C8  Ay..p. <..o...QH
020020  FF FC 4E 75 01 01 00 00  10 08 FF FF FF FF FF FF  .|Nu.....
020030  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF  .....
```

MVME117 1.x > M 20010;DI

Display executable code plus revision number, checksum, socket ID, and a few unused bytes following the routine.

```
020010  41F90001F000  LEA.L  $0001F000,A0 ?(CR)
020016  203C0000EFFF  MOVE.L  #61439,D0 ?(CR)
02001C  1100          MOVE.B  D0,-(A0) ?(CR)
02001E  51C8FFFC      DBF.L  D0,$02001C ?(CR)
020022  4E75          RTS  ?(CR)
020024  0101          BTST D0,D1 ?(CR)
                        0101 is revision.

020026  0000          DC.W  $0000 ?(CR)
                        0000 is where checksum is to be placed.

020028  1008          DC.W  $1008 ?(CR)
                        1008 are socket locations U16 and U08.

02002A  FFFF          DC.W  $FFFF ?(CR)
                        FFFF is unused memory.

02002C  FFFF          DC.W  $FFFF ?(CR)
                        FFFF is unused memory.

02002E  FFFF          DC.W  $FFFF ?(CR)
                        FFFF is unused memory.

020030  FFFF          DC.W  $FFFF ?(CR)
                        FFFF is unused memory.
```

CS

**EXAMPLE**

MVME117 1.x > CS 20000 2002A

Physical Address=00020000 0002002A  
(Even Odd)=4B34

MVME117 1.x > M 20026;W

020026 0000 ?4B34.

MVME117 1.x > CS

Physical Address=00020000 0002002A  
(Even Odd)=0000

MVME117 1.x > .R3 2000

MVME117 1.x > CS 0+R3 2A+R3

Physical Address=00020000 0002002A  
(Even Odd)=4B34

MVME117 1.x > M 26+R3;W

000026+R3 0000 ?4B34.

MVME117 1.x > CS

Physical Address=00020000 0002002A  
(Even Odd)=0000

MVME117 1.x &gt;

**COMMENT**

Request checksum of area using absolute addresses.

Checksum of even bytes is \$4B.  
Checksum of odd bytes is \$34.

Place these bytes in zeroed area used while calculating checksum.

Verify checksum (no operands needed if same as previous entries).  
Result is 0000, good checksum.

Define value of relative offset register 3.

Request checksum of area using relative offset.

Checksum of even bytes is \$4B.  
Checksum of odd bytes is \$34.

Place these bytes in zeroed area used while checksum was calculated.

Verify checksum (no operands needed if same as previous entries).

#### 4.2.14 Data Conversion (DC)

DC

DC &lt;expression&gt;

The DC command is used to convert an expression into hexadecimal and decimal. The expression may be entered in hexadecimal, decimal, or mixed format; output will be shown both ways. Default input format is hexadecimal. Octal and binary values may also be converted to decimal and hexadecimal values.

The following symbols are used:

\$	precedes hexadecimal value (default; may be omitted)
&	precedes decimal value
@	precedes octal value
%	precedes binary value

Except for .R0, offset registers may not be used with the DC command.

This command is useful in calculating displacements such as destination of relative branch instructions or program counter relative addressing modes.

##### COMMAND FORMAT

##### DESCRIPTION

MVME117 1.x > DC \$<data>

Convert hexadecimal data into hexadecimal and decimal.

MVME117 1.x > DC &<data>

Convert decimal data into hexadecimal and decimal.

##### EXAMPLE

MVME117 1.x > DC &120  
000078 =\$78=&120

MVME117 1.x > DC &15+\$4-\$13  
000000 =\$0=&0

MVME117 1.x > DC -1000  
FFF000 =\$FFFFFF000=-\$1000=-&4096

MVME117 1.x > DC &15-\$9+@14-%1100  
000006 =\$6=&6

MVME117 1.x &gt;

#### 4.2.15 Display Formatted Registers (DF)

**DF**
**DF**

The DF command is used to display the MC68010 registers. The registers display is also provided whenever the firmware monitor gains control of the program execution (i.e., at breakpoints and when tracing).

Note that any single register can be displayed with the .A0-.A7, .D0-.D7, and similar commands. Refer to the descriptions of the Display/Set Register command (.<register>) and the OF command.

##### EXAMPLE

##### COMMENT

MVME117 1.x > DF

Display formatted registers. Notice that the values of register A7 and the user stack pointer are the same because the status register indicates user mode.

```
PC=00000000 SR=0000=..0.... USP=FFFFFFF SSP=00000000 VBR=00000000 SFC=2 DFC=7
D0-7 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 00000000 00000000 00000000 00000000 00000000 00000000 00000000 FFFFFFFF
PC=000000 0000 DC.W $0000
```

MVME117 1.x > .A7 1100

MVME117 1.x > DF

Once again the values of A7 and the user stack pointer are the same. The latter was changed by altering A7 in the user mode.

```
PC=00000000 SR=0000=..0.... USP=00001100 SSP=00000000 VBR=00000000 SFC=2 DFC=7
D0-7 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00001100
PC=000000 0000 DC.W $0000
```

MVME117 1.x > .SS B00

Set supervisor stack pointer.

MVME117 1.x > .SR 2700

Set supervisor state and interrupt level 7.

MVME117 1.x > .R3 A00

Set relative offset register 3 to \$A00.

MVME117 1.x > .PC 0+R3

Set program counter to start of area using the relative offset register.

**DF**
**EXAMPLE**
**MVME117 1.x > DF**
**COMMENT**

Display formatted registers again. Notice that the supervisor mode in the status register results in the supervisor stack pointer being displayed both as the SSP and A7. Other changes include the program counter now being displayed in two ways: on the first line as an absolute address, and on the fourth line relative to the closest offset register equal to or below the absolute address. Notice also that the current location of the program counter is displayed in both hexadecimal and disassembled MC68010 source statements.

```
PC=00000A00 SR=2700=.S7.... USP=00001100 SSP=00000B00 VBR=00000000 SFC=2 DFC=7
D0-7 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
A0-7 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000B00
PC=000000+R3 41F81000          LEA.L    $00001000,A0
```

**MVME117 1.x >**

#### 4.2.16 Dump Memory (S-Records) (DU)

**DU**

DU[<port number>] <address1> <address2> [<text>]

The DU command formats memory data in S-record form and sends it to a specified port. The default port number is port 1, the MVME117 built-in CRT terminal/keyboard. The first record output is an S0 record, which will contain the characters entered in the text field on the command line, if any. The last record output is an S7, S8, or S9 terminator. Refer to Appendix D for information on S-records.

To dump to a peripheral using the DU command, a dual serial I/O module, MVME400, or a dual parallel I/O module, MVME410, must be available on the I/O Channel. Note that serial ports 1 and 2 on the MVME400 correspond to MVME117 ports 8 and 7, respectively, and that parallel ports 1 and 2 on the MVME410 correspond to MVME117 ports 9 and A, respectively.

Default destination is the console terminal. Specifying DU<port number> allows the output to be directed to another port.

Valid port numbers for this command are:

<u>PORT NUMBER</u>	<u>DESCRIPTION</u>
none	Default MVME117 port 1 (connector on MVME117)
1	Specifies MVME117 port 1 (connector on MVME117)
2	Specifies MVME117 port 2 (connector on MVME708-1 Transition Module)
3	Specifies MVME117 printer (connector on MVME708-1 Transition Module)
4	Specifies MVME050 terminal 1 (connector on MVME701 Transition Module)
5	Specifies MVME050 terminal 2 (connector on MVME701 Transition Module)
6	Specifies MVME050 printer (connector on MVME701 Transition Module)
7	Specifies MVME400 terminal 2 (connector on MVME400)
8	Specifies MVME400 terminal 1 (connector on MVME400)
9	Specifies MVME410 printer A (connector on MVME410)
A	Specifies MVME410 printer B (connector on MVME410)

This command does not send control characters to start or stop peripheral devices.

See also: LO, PF, VE

#### NOTE

Offset R0 is added to the address field in each S-record.

DU

EXAMPLE

COMMENT

MVME117 1.x > MD 1A00 30      Display memory where routine to be transferred exists.

```
001A00  41 F8 10 00 20 3C 00 00  02 FF 11 00 51 C8 FF FC  Ax.. <.....QH.:
001A10  60 EE 4E 71 4E 71 4E 71  4E 71 4E 71 4E 71 4E 71  'nNqNqNqNqNqNqNq
001A20  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
```

MVME117 1.x > MD 1A00 7;DI      Display memory using the disassembler.

```
001A00  41F81000      LEA.L   $00001000,A0
001A04  203C000002FF  MOVE.L  #767,D0
001A0A  1100          MOVE.B  D0,-(A0)
001A0C  51C8FFFC      DBF.L   D0,$001A0A
001A10  60EE          BRA.S   $001A00
001A12  4E71          NOP
001A14  4E71          NOP
```

MVME117 1.x > DU 1A00 1A15 MVME117 Test

Dump memory to default port, starting at \$1A00 through \$1A15, and use ASCII text in S0 record.

```
Physical Address=00000A00 00000A15
S012000054454E42554720322E582054455354F0
S1130A0041F81000203C000002FF110051C8FFFC17
S1090A1060EE4E714E7110
S9030000FC
```

NOTE: Default port is the terminal connected to the MVME117 RS-232C connector.

MVME117 1.x >

## 4.2.17 Go Direct Execute Program (GD)

**GD**

GD [<address>]

The GD command is similar to the GO command, except that GD does not set breakpoints, nor does it start by tracing one instruction. The GD command starts the target program at the location given as <address> without changing any of the exception vectors (default locations 0 through \$3FF). If <address> is not specified, the GD command starts the target program at the address in the program counter.

See also: GO, GT

### EXAMPLE

### COMMENT

```
(Listing of program in memory at location 001900)
001900  1018      MOVE.B  (A0)+,D0
001902  0C000000  CMP.B   #0,D0
001906  66F8      BNE.S   $001900
001908  4E75      RTS
```

MVME117 1.x > BR 1900 1908

Breakpoints

```
001900  001900
001908  001908
```

MVME117 1.x > G 1900  
Physical Address=00001900

GO inserts breakpoints, traces one instruction, and hits breakpoint at \$1900.

At Breakpoint

```
PC=00001900 SR=2704=.S7..Z. USP=0000C19E SSP=00000C00 VBR=00000000 SFC=2 DFC=2
D0-7 00000000 00000000 00003048 4D453455 00000000 00000020 00000000 0007FFFE
A0-7 00001002 00001694 0000065C 00F01D72 00F0120C 00000538 00000538 00000C00
PC=001900
```

MVME117 1.x > GD 1900  
Physical Address=00001900

GD does not insert breakpoints but simply transfers control to the target routine after loading its registers.

**4.2.18 Go Execute Program (GO)**GO  
GGO [<address>]  
G [<address>]

The G (or GO) command causes the target registers (previously saved in RAM) to be placed into the actual MC68010 hardware registers, and any breakpoints previously requested to be placed into RAM. When this is completed, control is given to the target program by one of two methods. If no operands are provided with the G (or GO) command, the current value of the program counter is used. If an address is provided, this address will be placed into the program counter and then used to give control to the target code. The program starts by first tracing one instruction and then free running until one of the following events interrupts the program execution.

- a. The target program encounters a breakpoint.
- b. An abnormal program sequence causes exception processing (e.g., divide by zero).
- c. The operator intervenes through use of the RESET or ABORT switches on the MVME117 operator panel.

**NOTE**

The execution will be in real-time unless any breakpoints with <count> are encountered.

The [<address>] parameter can be provided in several formats:

- a. An absolute address (24-bit maximum).
- b. An expression using a displacement + relative offset register.
- c. Address indirect, using the contents of RAM (or ROM) to acquire the new program counter contents.
- d. Register indirect, using the contents of address registers 0 through 7 to acquire the new program counter contents.

GO  
G

### EXAMPLE

MVME117 1.x > .PC 2A00

MVME117 1.x > G

Physical Address=00002A00

At Breakpoint

PC=002A0A 1100 MOVE.B D0,-(A0)

MVME117 1.x > G 2A00

Physical Address=00002A00

At Breakpoint

PC=002A0A 1100 MOVE.B D0,-(A0)

MVME117 1.x > M 20000;L

020000 00000000 ?2A00.

MVME117 1.x > GO [20000]

Physical Address=00002A00

At Breakpoint

PC=002A0A 1100 MOVE.B D0,-(A0)

MVME117 1.x > .A1 2A00

MVME117 1.x > G (A1)

Physical Address=00002A00

At Breakpoint

PC=002A0A 1100 MOVE.B D0,-(A0)

MVME117 1.x > .R2 2A00

MVME117 1.x > GO 0+R2

Physical Address=00002A00

At Breakpoint

PC=00000A+R2 1100 MOVE.B D0,-(A0)

MVME117 1.x >

### COMMENT

Set program counter to desired address.

Enter GO command using existing PC.

Enter GO command with absolute address provided.

Set RAM location to contain an execution address.

Enter GO command providing indirect address in RAM.

Set address register to contain an execution address.

Enter GO command providing indirect addressing in A1.

Set relative offset register to contain start of module.

Enter GO command providing a displacement and offset register.

Notice the physical address used in each example is identical, though provided in a different way in each case.

**4.2.19 Go Until Breakpoint (GT)**

GT

GT &lt;temporary breakpoint address&gt;

The GT command performs the following:

- a. Sets the temporary breakpoint specified on the command line.
- b. Sets breakpoints entered by the BR command.
- c. Sets target program registers as displayed by the DF command.
- d. Causes the target program to execute from the PC address (free run in real-time).

When any breakpoint is encountered, the temporary breakpoint is removed.

**NOTE**

If the same address is specified as an operand in the GT command as previously entered with a BR command, "ERROR" is displayed on the terminal.

See also: BR, DF, GD, GO, TR, TT

**EXAMPLE**

(Listing of program in memory at location 002900)

002900	1018	MOVE.B	(A0)+,D0
002902	0C000000	CMP.B	#0,D0
002906	66F8	BNE.S	\$002900
002908	4E75	RTS	

MVME117 1.x > BR 2900 2908

Breakpoints

002900	002900
002908	002908

MVME117 1.x > .PC 2900MVME117 1.x > GT 2906

Physical Address=00002906

Physical Address=00002900

GT

EXAMPLE

At Breakpoint

PC=00002906 SR=2700=.S7.... USP=0000C19E SSP=00000BF8 VBR=00000000 SFC=2 DFC=2  
D0-7 00000020 00000000 00003048 4D453455 00000000 00000020 00000000 0007FFFE  
A0-7 0000260E 00002694 0000065C 00F01D72 00F0120C 00000538 00000538 00000BF8  
PC=002906

MVME117 1.x > BR

Breakpoints

002900 002900  
002908 002908

MVME117 1.x &gt;

#### 4.2.20 Help (HE)

HE

HE

The HE command displays a list of available commands, status of Power-Up Test, and Autoboot, as well as a list of port assignments.

#### EXAMPLE

MVME117 1.x > HE

POWER-UP test ENABLED

AUTOBOOT from Drive 3 Controller 0 DISABLED

```
.PC .SR .US .SS .VBR .DFC .SFC
.DO - .D7
.A0 - .A7
.R0 - .R7
```

CMOS	IOC	IOP	IOT	RESET	SET	TIME						
AB	NOAB	BD	BF	BH	BI	BM	BO	BR	NOBR	BS	BT	
CS	DC	DF	DU	G	GD	GO	GT	HE	LO	M	MD	
MM	MS	OF	PA	NOPA	PF	PT	NOPT	ST	T	TA	TM	
TR	TT	VE	VI									

#### -PORTS-

```
MVME117 MVME050 MVME400 MVME410
1=Term1 4=Term1 7=Term2 9=PRT A
2=Term2 5=Term2 8=Term1 A=PRT B
3=PRT 6=PRT
```

#### -Disk Controllers-

```
SCSI=0 thru 7
RWIN=8
```

MVME117 1.x &gt;

#### 4.2.21 I/O Command for Drive/Controller (IOC)

IOC

##### IOC

IOC allows commands to be issued directly to a specific controller and drive. Both SCSI and RWIN controllers are supported with their very different style of command packets. When invoked, this command prompts for a drive and controller, and by examining the controller number, IOC selects either the SCSI (0-7), or RWIN (8) controller, and displays both the address and contents of the corresponding packet. The information displayed reflects the last command issued and status received by that controller.

After displaying the contents of the last command packet, an "Are you sure" prompt is presented, allowing the user to either bypass or issue the packet as it currently exists. A typical sequence would include an initial IOC for the specific drive and controller just to see where the packet is located. In response to the "Are you sure" prompt, an "N" is entered. Then the particular command or data parameter is modified within the packet via Memory Modify (MM), and a subsequent IOC entered. This time a "Y" is entered at the "Are you sure" prompt. If the command was completed without error, the MVME117 prompt is displayed. Otherwise, the controller specific completion code or status bytes are displayed along with the current pseudo registers.

This command is used primarily as a debugging tool to issue commands to a controller in an attempt to locate problems with either drives, media, or the controller itself. For more information about SCSI command packets and their use, refer to Appendix E, SCSI Firmware Support, in this manual. For more information about the RWIN controller and its command set, see the Winchester Disk Controller User's Manual.

While answering the prompts, there are four actions that can be taken following the question mark prompt:

- ? (CR) -- Entering a carriage return indicates that the existing value for the current parameter is acceptable; go on to the next parameter.
- ? . -- Entering a period indicates that this execution of the IOC command must be terminated now, without asking for more parameters.
- ? ^ -- Entering a caret symbol indicates that a previous parameter requires a change and logically backs up one parameter each time it is entered (until the first entry is reached, where it remains until one of the other responses is received).
- ? <data> -- Entering the appropriate data requested (followed by a carriage return or ENTER). Often the parameters are checked for valid options (i.e., Y or N).

**IOC**

A list of commands is provided here for each controller type:

<u>SCSI Function</u>	<u>Description</u>
\$00	Read
\$04	Write
\$08	Attach
\$0C	Detach
\$10	Format (with/without defect list)
\$14	Assign alternate sector (SCSI)
\$18	Assign alternate track (SASI)
\$1C	Custom SCSI sequence
\$20	SCSI bus reset
\$24	SCSI controller reset

<u>RWIN Function</u>	<u>Description</u>
\$00	Check drive status
\$01	Recalibrate
\$04	Format drive
\$06	Format track (refer to example)
\$07	Format default/alternate track
\$08	Read sectors (RWIN commands requiring data transfers must use IOP)
\$09	Scan sectors
\$0A	Write sectors (RWIN commands requiring data transfers must use IOP)
\$0B	Seek
\$0D	Read ECC (Error Correction Code is supported on Winchester drives only)
\$0E	Write ECC (Error Correction Code is supported on Winchester drives only)
\$60	Configure drive

The following example illustrates a typical IOC sequence that interfaces with the SCSI controller, first to attach a controller/drive and then to read one sector into RAM.

EXAMPLE
COMMENT

```

MVME117  1.x > IOC
          Drive # =.....$00 ? 2
          Controller # =.....$00 ? (CR)

          Packet address= $00000E00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
No previous SCSI I/O has
taken place since the command
packet is still zeros.

Are you sure, (Y/N) ? N

```

Do not issue this packet as a command.

IOC

**EXAMPLE**

 MVME117 1.x > .R1 E00

 MVME117 1.x > M 0+R1;W

 000000+R1 0000 ?2

 000002+R1 0000 ?(CR)

 000004+R1 0000 ?FF

 000006+R1 0000 ?2

 000008+R1 0000 ?50

 00000A+R1 0000 ?28

 00000C+R1 0000 ?10

 00000E+R1 0000 ?(CR)

 000010+R1 0000 ?7

 000012+R1 0000 ?(CR)

 000014+R1 0000 ?8

 000016+R1 0000 ?46

 000018+R1 0000 ?(CR)

 00001A+R1 0000 ?2005

 00001C+R1 0000 ?100

 00001E+R1 0000 ?(CR)

 000020+R1 0000 ?(CR)

 000022+R1 0000 ?D22.

 MVME117 1.x > IOC

 Drive # =.....\$02 ? (CR)

 Controller # =.....\$00 ? (CR)

Packet address= \$00000E00

00 02 00 00 00 FF 00 02 00 50 00 28 00 10 00 00 issued).

00 07 00 00 00 08 00 46 00 00 20 05 01 00 00 00

00 00 0D 22 00 00 00 00 00 00 00 00 00 00 00 00

 Are you sure, (Y/N) ? Y

 MVME117 1.x > M 0+R1;W
**COMMENT**

Set offset register number 1 to packet address, allowing memory offsets to match the packet descriptions defined in Appendix E. The following data is provided to build an attach packet for drive 2 and controller 0.

Controller LUN 0 and device LUN 2.  
Status bytes 0 and 1.

Stepping rate (approx. 12 ms).

Number of heads.

Number of cylinders.

Precompensation cylinder.

Sectors per track.

Reserved.

SCSI drive attributes.

Reserved.

SCSI "attach" function code.

Vector to use to return.

Status bytes 2 and 3.

Controller type and retry count.

Sector size.

Reserved.

Address of

160-byte RAM area.

Now that packet is built, request IOC for drive 2 controller 0. The packet has been built to attach the drive and controller, (a requirement before the read can be issued).

This time issue command packet.

Now change the packet to become a read. Note that the SCSI command completed normally, as indicated by two observations: (1) The MVME117 prompt was displayed immediately following the "Are you sure" response. (If an error had occurred, the SCSI error message would have been displayed before the prompt.); (2) The second word in the packet contains \$2000 indicating that the "Attach" completed normally.

**IOC**
**EXAMPLE**

```

000000+R1 0002 ?(CR)
000002+R1 2000 ?0
000004+R1 00FF ?0
000006+R1 0002 ?2000
000008+R1 0050 ?0
00000A+R1 0028 ?0
00000C+R1 0010 ?1
00000E+R1 0000 ?(CR)
000010+R1 0007 ?0
000012+R1 0000 ?(CR)
000014+R1 0008 ?0
000016+R1 0046 ?(CR)
000018+R1 0000 ?_
MVME117 1.x > BF 2000 3000 4161
Physical Address=00002000 00003000
MVME117 1.x > IOC
      Drive # =.....$02 ? (CR)
      Controller # =.....$00 ? (CR)

```

Packet address= \$00000E00

```

00 02 00 00 00 00 20 00 00 00 00 00 00 01 00 00
00 00 00 00 00 00 00 46 00 00 20 05 01 00 00 00
00 00 0D 22 00 00 00 00 00 00 00 00 00 00 00 00

```

Are you sure, (Y/N) ? Y

MVME117 1.x > R1 0

MVME117 1.x > MD 2000 90

```

002000 54 53 31 35 00 00 00 00 00 02 00 02 00 00 00 04 TS15.....
002010 00 00 00 00 00 00 00 0A 00 04 00 00 00 00 00 00 .....
002020 20 00 00 00 08 3B 54 45 53 54 20 54 52 41 50 20 ....;TEST TRAP
002030 23 31 35 20 53 43 53 49 20 20 30 34 32 30 E9 C7 #15 SCSI 0420iG
002040 0F 1E 2D 3C 4B 5A 69 78 87 96 A5 B4 C3 D2 E1 F0 ..-<KZix..%4CRap
002050 0F 1E 2D 3C 4B 5A 69 78 87 96 A5 B4 C3 D2 E1 F0 ..-<KZix..%4CRap
002060 F1 F2 F4 F8 F9 FA FC FE FF 7F BF DF EF 6F AF CF qrtxyz|~..?_oo/O
002070 4F 8F 0F 07 0B 0D 0E 06 0A 0C 04 08 04 02 01 00 0.....
002080 41 61 41 61 41 61 41 61 41 61 41 61 41 61 41 61 AaAaAaAaAaAaAaAa
MVME117 1.x >

```

**COMMENT**

Controller LUN 0 and device LUN 2.  
Clear status bytes 0 and 1.

Memory address \$2000.

Starting sector number 0.  
Number of sectors = 1.

SCSI "read" function.  
Vector to use to return.

Place known pattern (AaAa)  
into RAM.  
Now that the drive and controller  
are attached, use IOC to perform  
the read.

Yes, packet is ready for the read.  
Zero offset register number 1.  
Display first \$90 bytes of data.

Notice that a physical sector of  
\$80 bytes was read containing the  
first half of the volume ID. Also  
notice that the pattern placed  
into RAM is still there following  
the one-sector read.

**IOC**

The following RWIN example shows how a "Format Track" command can be issued to the RWIN controller using IOC.

**EXAMPLE**
**COMMENT**

MVME117 1.x > IOC

Invoke the physical disk I/O command.

Read/Write(R/W)=.....R ? W  
 Memory Address =.\$00001000 ? 2000  
 Drive # =.....\$00 ? 2  
 Controller # =.....\$00 ? 8  
 First BLOCK # =.\$00000000 ? 100  
 # of Blocks =.....\$0000 ? 1

Specify a write operation.  
 Write to memory location \$2000.  
 Write to drive 2 (floppy).  
 Write to RWIN controller.  
 Write to block number \$100.  
 Write one 256-byte block.

Are you sure (Y/N) ? Y

Last chance ... Sure? Yes

DISK ERROR: STATUS=06 4D 00 08 09 01 40 01 00 09

For this example an unformatted floppy disk was placed into drive 2. As expected an "06" indicates that the ID HEADER NOT FOUND message from the Winchester Disk Controller User's Manual is the correct description of the situation.

PC=00F04EBC SR=2700=.S7.... USP=FFFFFFFF SSP=000008CC VBR=00000000 SFC=2 DFC=7  
 D0-7 00000001 000000AC 00002100 00000109 00000000 00000010 00000028 00042700  
 A0-7 00F1C0D1 00002100 00000682 00F1C0D9 00001010 0000054E 0000054E 000008CC  
 PC=F04EBC 4BFAFDC0 LEA.L \$00F04C7E(PC),A5

MVME117 1.x > IOC

The IOC command will be used to write the ID header for a specific track. To start, the RWIN command must be located. This is done by entering an IOC command. The RWIN command last used is then displayed along with its address. In this case, the last request was "0A" for the write that was attempted in the IOP.

Drive # =.....\$02 ? (CR)  
 Controller # =.....\$08 ? (CR)

Packet(s) at \$00000546 0A 40 01 08 01 AC

Are you sure, (Y/N) ? N

**IOC**
**EXAMPLE**

MVME117 1.x > M 546

000546 0A ?06

**COMMENT**

Modify memory at the specified address (\$546 in this example) and change the write (\$0A) operation to a format track (\$06) operation. (Chapter 4 in the Winchester Controller User's Manual.)

4

MVME117 1.x > IOC

Drive # = .....\$02 ? (CR)

Controller # = .....\$08 ? (CR)

Invoke the IOC command and, after verifying the DRIVE correct drive and controller and that the RWIN command is proper, reply Y in response to the "Are you sure?" prompt.

Packet(s) at \$00000546 06 40 01 08 01 AC

Are you sure, (Y/N) ? Y

MVME117 1.x > IOP

Read/Write(R/W)=.....W ? (CR)

Memory Address =.\$00002000 ? (CR)

Drive # =.....\$02 ? (CR)

Controller # =.....\$08 ? (CR)

First BLOCK # =.\$00000100 ? (CR)

# of Blocks =.....\$0001 ? (CR)

With the return of the MVME117 prompt the command is complete. Note that a total format will require longer than the "dead man" timer provides; the MVME117 program will time out and send an error message, but the RWIN controller will proceed with the command until complete.

Are you sure (Y/N) ? Y

"W" Ok

In this example the write can now be completed, as long as the one track that was formatted is all that is specified.

MVME117 1.x >

## IOP

**4.2.22 I/O Physical for Drive/Controller (IOP)**

## IOP

The IOP command allows the user to do logical reads or writes of 256-byte blocks to or from the disk. When invoked, this command prompts for the information required to perform the input/output operation. The initial values for drive and controller are zeros, (hard disk 0 on the SCSI controller 0). However, once a parameter has been changed the new value will become the default.

Notice that this command can only perform reads and writes. If for some reason any of the other commands are required, the IOC command can be used.

While answering the prompts, there are four actions that can be taken following the question mark prompt:

- ? (CR) -- Entering a carriage return indicates that the existing value for the current parameter is acceptable; go on to the next parameter.
- ? . -- Entering a period indicates that this execution of the IOC command must be terminated now, without asking for more parameters.
- ? ^ -- Entering a caret symbol indicates that a previous parameter requires a change and will logically back up one parameter each time it is generated (until the first entry is reached, where it will remain until one of the other responses is received).
- ? <data> -- Entering the appropriate data requested (followed by a carriage return or ENTER). Often the parameters are checked for valid options (i.e., Y or N).

**CAUTION**

CARE SHOULD BE TAKEN WHEN USING THIS DIAGNOSTIC TOOL. PORTIONS OF THE OPERATING SYSTEM COULD BE DESTROYED IF AN INCORRECT AREA OF A DISK WERE MODIFIED.



IOP

**EXAMPLE**

Are you sure, (Y/N) ? Y

SCSI Disk Error: 20120000

```
PC=00F02D96 SR=2200=.S2.....USP=FFFFFFF
DO-7 00000118 00000081 00000007 00000001
AO-7 00000D5E 00F017FA 00000E00 00000000
PC=F02D96 42380E84 CLR.B $00000E84
MVME117 1.x >
```

**COMMENT**

The SCSI disk error data is extracted from the two words of status in the packet. \$2012 indicates a selection time-out. (Refer to Appendix E for complete SCSI error codes.)

```
SSP=00000F00 VBR=00000000 SFC=0 DFC=0
38303130 A987EDCB 00000022 00002012
00F01834 00000570 00F58001 00000F00
```

The next examples are for a Winchester (RWIN) controller.

MVME117 1.x > IOP

Read/Write(R/W)=.....R ? (CR)

Memory Address =.\$00000000 ? 2000

Drive # =.....00 ? ^

Memory Address =.\$00002000 ? 2A00

Drive # =.....\$00 ? 2

Controller # =.....\$00 ? 8

First BLOCK # =.\$00000000 ? 500

# 256 Byte Blocks=.....\$0000 ? 1

Request disk I/O to read a routine

Use default of read.

Change \$0 default to \$2000.

That isn't correct; back up one.

Change \$2000 to \$2A00.

Change default drive to drive 2.

Use controller 8. (RWIN)

Change block number to \$500.

Read one 256-byte block.

Are you sure? (Y/N) ? Y

Last chance? Yes, all is ready.

"R" Ok

The read is complete.

MVME117 1.x > M 2A12;L

002A12 4E714E71 ? -1

002A16 4E714E71 ? -1

002A1A 45C25380 ? -1

002A1E EB080206 ? -1

Change RAM to where routine was loaded.

MVME117 1.x > IOP

Read/Write(R/W)=.....R ? W

Memory Address =.\$00002A00 ? (CR)

Drive # =.....\$02 ? (CR)

Controller # =.....\$08 ? (CR)

First BLOCK # =.\$00000500 ? (CR)

# 256 Byte Blocks=.....\$0001 ? (CR)

Are you sure (Y/N) ? Y

Request disk I/O to write to a disk.

Change to W for write.

Use previous address.

Drive is fine; no change.

No change.

No change.

No change.

Last chance; are you sure?

Yes, all is ready.

"W" Ok

Write is complete.

MVME117 1.x &gt;

#### 4.2.23 I/O Teach for a Drive/Controller (IOT)

IOT

IOT [&lt;device&gt;][&lt;controller&gt;]

where:

<device> is a single hexadecimal digit, 0 through 7, specifying the disk to be read. Default is 0 or previous drive value.

<controller> is a single hexadecimal digit, 0 through 8, specifying the controller through which the disk is connected. Default is 0 or previous controller value.

**4**

The IOT command allows the user to change the configuration of the controller/drive. If the IOT command is invoked without specifying <device> or <controller>, the command will prompt for required information. If the <device> and/or <controller> are specified in the IOT command line, the current configuration is overwritten with the configuration data located on the media, without the user having to know and manually enter the parameter information required. When a SCSI controller number is specified (controller number 0 - 7), the current local level of the MVME117 is displayed, followed by a prompt to allow its modification.

The parameters required for correct configuration when the options are not specified depend upon the type of drive and controller, as shown below:

WINCHESTER HARD DISK

Drive number  
Controller number  
Sector size  
Number of heads  
Number of cylinders  
Number of sectors per track

5-1/4 INCH FLOPPY DISK

Drive number  
Controller number  
Sector size  
Number of heads  
Number of cylinders  
Number of sectors per track  
Motorola/IBM format  
Single- or double-sided media  
Single- or double-track density  
Single- or double-data density

## IOT

The IOT command will present the appropriate questions based upon which drive has been specified. There are four actions that can be taken following a question mark prompt:

- ? (CR) -- Entering a carriage return indicates that the existing value for the current parameter is acceptable; go on to the next parameter.
- ? . -- Entering a period indicates that this execution of the IOT command must be terminated now, without asking for more parameters.
- ? ^ -- Entering a caret symbol indicates that a previous parameter requires a change and logically backs up one parameter each time it is entered (until the first entry is reached, where it will remain until one of the other responses is received).
- ? <data> -- Entering the appropriate data requested (followed by a carriage return or ENTER). Often the parameters are checked for valid options (i.e, Y or N).

Appropriate configuration information for specific disk types is listed in the SCSI "attach" packet description contained within Appendix E in this manual, or in the "Mass Storage" chapter of the VERSAdos to VME Hardware and Software Configuration User's Manual.

**IOT**

The examples on this page are for a SCSI controller; those on the next page are for an RWIN controller.

**EXAMPLE**
**COMMENT**

```
MVME117 1.x > IOT 0,0
      Drive # =.....$00 ? (CR)
      Controller # =.....$00 ? (CR)
      Controller Type=SCSI
      2=DTC520-DB
      3=DTC520-B
      8=Saber-AP
      Controller Code=.....$ 3 ? (CR)
      Media Type=Hard
      $80
      $100
      $200
      $400
      Sector Size=.....$0100 ? (CR)
      Number of Heads=.....$06 ? (CR)
      No. of Cylinders=.....$0132 ? (CR)
      Sectors/Track=.....$21 ? (CR)
MVME117 Lvl.(0-7)=.....7 ? (CR)
MVME117 1.x >
```

Configure and display SCSI drive 0 and controller 0 from the media. CR was entered at each prompt.

These parameters are for a DTC520-B controller with a 15 Megabyte hard disk.

SCSI controllers prompt for the MVME117 local level -- 0 through 7.

```
MVME117 1.x > IOT 2,0
      Drive # =.....$02 ? (CR)
      Controller # =.....$00 ? (CR)
      Controller Type=SCSI
      2=DTC520-DB
      3=DTC520-B
      8=Saber-AP
      Controller Code=.....$ 3 ? (CR)
      Media Type=5" FLOPPY
      $80
      $100
      $200
      $400
      Sector Size=.....$0100 ? (CR)
      Number of Heads=.....$02 ? (CR)
      No. of Cylinders=.....$0050 ? (CR)
      Sectors/Track=.....$10 ? (CR)
      Sides(S/D)=.....D ? (CR)
      TRK-Density(S/D)=.....D ? (CR)
      DATA Density(S/D)=.....D ? (CR)
      IBM Format (Y/N)=.....Y ? (CR)
MVME117 Lvl.(0-7)=.....7 ? (CR)
MVME117 1.x >
```

Configure and display SCSI drive 2 and controller 0 from the media. Again CR was entered at each prompt.

Drives 0 and 1 are hard disks, drives 2 and 3 are 5-1/4 inch floppy drives for a DTC 520-B controller.

**IOT**
**EXAMPLE**

```
MVME117 1.x > IOT
          Drive # =.....$00 ? 3
          Controller # =.....$08 ? (CR)
          Controller Type=RWIN
0= 128
1= 256
2= 512
3=1024
  Sector Size(0-3)=..... 1 ? (CR)
  Number of Heads=.....$02 ? (CR)
  No. of Cylinders=.....$0050 ? (CR)
  Sectors/Track=.....$10 ? (CR)
  Format (M/I)=.....I ? (CR)
  Sides(S/D)=.....D ? (CR)
  TRK Density (S/D)=.....D ? (CR)
  DATA Density (S/D)=.....D ? (CR)
```

```
MVME117 1.x > IOT 3,8
          Drive # =.....$03 ? (CR)
          Controller # =.....$08 ? (CR)
          Controller Type=RWIN
0= 128
1= 256
2= 512
3=1024
  Sector Size(0-3)=..... 1 ? (CR)
  Number of Heads=.....$02 ? (CR)
  No. of Cylinders=.....$0050 ? (CR)
  Sectors/Track=.....$10 ? (CR)
  Format (M/I)=.....I ? (CR)
  Sides (S/D)=.....D ? (CR)
  TRK Density (S/D)=.....D ? (CR)
  DATA Density (S/D)=.....D ? (CR)
```

```
MVME117 1.x > IOT 0,8
          Drive # =.....$02 ? 00
          Controller # =.....$08 ? (CR)
          Controller Type=RWIN
0= 128
1= 256
2= 512
3=1024
  Sector Size(0-3)=..... 1 ? (CR)
  Number of Heads=.....$06 ? (CR)
  No. of Cylinders=.....$033E ? (CR)
  Sectors/Track=.....$20 ? (CR)
MVME117 1.x >
```

**COMMENT**

Change configuration.  
Select drive 3.  
#8 is default from a previous IOT.  
Verifies controller type is RWIN.  
Menu shows codes and appropriate sector sizes.

Display RWIN, floppy media config.

Display RWIN, hard disk config data.

NOTE: These parameters reflect the configuration for a 15 Megabyte drive.

#### 4.2.24 Load (S-Records) (L0)

L0

```
L0[<port number>] [;<options>] =<text>
```

The L0 command prepares the MVME117 to receive S-records from the designated <port number> and then transmits the <text> following the = sign to the system connected to the <port number> indicated. As the S-records are received, the checksums are verified and the data placed into memory. If the automatic relative offset register, R0, contains a nonzero value, this offset is added to the address contained within the S-record before the data is moved into memory.

The following options are supported:

- ;-C Ignore validation of the checksum on each S-record while loading.
- ;X Echo the S-records read to the console. Different environments may dictate that the ;X option not be used. If printer attach is in effect, the data cannot be displayed upon the screen (and then printed on the printer) before the next record arrives. Thus data can be missed.

Default source is the console. Specifying L0<port number> allows the input to be received from other ports.

Valid port numbers for this command are:

<u>PORT NUMBER</u>	<u>DESCRIPTION</u>
none	Default MVME117 port 1 (connector on MVME117)
1	Specifies MVME117 port 1 (connector on MVME117)
2	Specifies MVME117 port 2 (connector on MVME708-1 Transition Module)
4	Specifies MVME050 terminal 1 (connector on MVME701 Transition Module)
5	Specifies MVME050 terminal 2 (connector on MVME701 Transition Module)
7	Specifies MVME400 terminal 2 (connector on MVME400)
8	Specifies MVME400 terminal 1 (connector on MVME400)

LO

**EXAMPLE**
**COMMENT**

MVME117 1.x > BF 2A00 2F00 2020  
Physical Address=00002A00 00002F00

Fill RAM with spaces.

MVME117 1.x > MD 2B00

Display RAM.

002B00 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20

MVME117 1.x > .RO 100

Set automatic relative offset register to \$100 to reposition S-records to be loaded by + \$100.

MVME117 1.x > LO =DU 2A00 2A80  
DU 2A00 2A80

Load S-records from \$2A00 to \$2A80 (into \$2B00 to \$2B80).

MVME117 1.x > .RO 0+R7

Reset automatic relative offset register to zero.

MVME117 1.x > .RO  
.RO=00000000

MVME117 1.x > MD 2A00

Display memory at \$2A00 to verify S-records not loaded here.

002A00 20 20 20 20 20 20 20 20 20 20 20 20 20 20

MVME117 1.x > MD 2B00 7F

Display memory at offset \$100 from source location.

```

002B00 00 01 02 03 04 05 FF FF 08 09 0A 0B 0C 0D 0E 0F .....
002B10 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F .....
002B20 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F  !"#$%&'()*+,-./
002B30 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 0123456789:;<=>?
002B40 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F @ABCDEFGHIJKLMNO
002B50 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F PQRSTUVWXYZ[\]^_
002B60 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 'abcdefghijklmnopqrstuvwxyz{|}~.
002B70 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F

```

**L0**
**EXAMPLE**
**MVME117 1.x > L0 ;X=DU 2A00 2A80**
**COMMENT**

Enter Load command specifying ;X option (echo S-records to CRT as memory is being loaded).

**DU 2A00 2A80**
**Physical Address=00002A00 00002A80**

Notice S0, S1, and S9 records are displayed upon screen. (Refer to warning in command description about timing restrictions with the ;X option.)

**S0030000FC**
**S1132A00000102030405FFFF08090A0B0C0D0E0F4A**
**S1132A10101112131415161718191A1B1C1D1E1F3A**
**S1132A20202122232425262728292A2B2C2D2E2F2A**
**S1132A30303132333435363738393A3B3C3D3E3F1A**
**S1132A40404142434445464748494A4B4C4D4E4F0A**
**S1132A50505152535455565758595A5B5C5D5E5F5A**
**S1132A60606162636465666768696A6B6C6D6E6F6A**
**S1132A70707172737475767778797A7B7C7D7E7F6A**
**S1042A8080D1**
**S90300000FC**
**MVME117 1.x > MD 2A00 80**

Display memory containing downloaded data. R0 was set to zero so S-records were loaded with zero offset.

002A00	00	01	02	03	04	05	FF	FF	08	09	0A	0B	0C	0D	0E	0F	.....
002A10	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	.....
002A20	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	!"#\$%&'()*+,-./
002A30	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F	0123456789:;<=>?
002A40	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	@ABCDEFGHIJKLMNO
002A50	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	PQRSTUVWXYZ[\]^_
002A60	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	'abcdefghijklmno
002A70	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F	pqrstuvwxyz{ }~.

**NOTE**

The host system used to create and transmit the S-records was an MC68000 Educational Computer Board (MEX68KECB).

#### 4.2.25 Memory Display (MD)

MD

```
MD[<port number>] <address> [<count>][;<options>]
```

The MD command displays a portion of memory which begins at <address> and extends for the number of bytes or lines given as <count>. There are two formats that can be requested with the MD command.

- a. The dump format begins each line with the starting or next hexadecimal memory address followed by 16 hex bytes per line with the ASCII equivalent shown to the right. The number of lines varies with the <count> entered (or default). There are no partial lines. If the byte count ends in the middle of a line, the complete line is displayed. (Default byte <count> is \$10.)
- b. The disassembler format provides:
  1. The starting or next hexadecimal memory address.
  2. The object code displayed in hexadecimal.
  3. The M68010 source statement that will assemble into the object code as described in 2. above.

If the operation code is not valid, a "Define Constant" is constructed for one word. Notice that <count> for the disassembler mode is a number of source lines to be disassembled and displayed, not the number of bytes. (Default line <count> is \$10.)

Default destination is the console terminal. Specifying MD<port number> allows the output to be directed to another port.

Valid port numbers for this command are:

<u>PORT NUMBER</u>	<u>DESCRIPTION</u>
none	Default MVME117 port 1 (connector on MVME117)
1	Specifies MVME117 port 1 (connector on MVME117)
2	Specifies MVME117 port 2 (connector on MVME708-1 Transition Module)
3	Specifies MVME117 printer (connector on MVME708-1 Transition Module)
4	Specifies MVME050 terminal 1 (connector on MVME701 Transition Module)
5	Specifies MVME050 terminal 2 (connector on MVME701 Transition Module)
6	Specifies MVME050 printer (connector on MVME701 Transition Module)
7	Specifies MVME400 terminal 2 (connector on MVME400)
8	Specifies MVME400 terminal 1 (connector on MVME400)
9	Specifies MVME410 printer A (connector on MVME410)
A	Specifies MVME410 printer B (connector on MVME410)

**MD**

Options supported are the disassembler and the screen option.

- ;DI** Requests the disassembler option. The <count>, if provided, is a line count (default is \$10).
- ;S** Requests the display of a full screen of memory (16 lines of display in either dump or disassembler format). Notice that the default for disassembly is \$10 (or 16 decimal) anyway. If the <count> and ;S option are both entered within the same MD command, the ;S option has priority.

All combinations are valid (e.g., ;DIS, ;SDI, ;S DI, ;DI S).

The MD command has a quick scroll facility that lets the terminal operator press (CR) repeatedly following the initial MD command. In the past, all but the first display were automatically 16 lines long. To enable control blocks to be examined conveniently, the <count> (either bytes or lines) is used for each iteration.

#### EXAMPLE

#### COMMENT

MVME117 1.x > MD 10000 30 Display memory of a small routine in dump format.

```
010000 41 F8 10 00 20 3C 00 00 02 FF 11 00 51 C8 FF FC Ax.. <.....QH.|
010010 60 EE FF FF FF FF FF FF FF FF FF FF FF FF 'n.....
010020 FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
```

MVME117 1.x > MD 10000 7;DI Display memory of the same routine in disassembler format.

```
010000 41F81000 LEA.L $00001000,A0
010004 203C000002FF MOVE.L #767,D0
01000A 1100 MOVE.B D0,-(A0)
01000C 51C8FFFC DBF.L D0,$01000A
010010 60EE BRA.S $010000
010012 FFFF DC.W $FFFF
010014 FFFF DC.W $FFFF
```

MVME117 1.x > MD 2A00 Display memory without a <count>.

```
002A00 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F .....
```

MVME117 1.x > (CR) Enter a carriage return; the next <count> bytes are displayed (default is \$10).

```
002A10 10 11 12 13 14 15 16 17 00 01 02 03 04 05 06 07 .....
```

MD

EXAMPLE
COMMENT

MVME117 1.x > MD 2A00;S      Display a full screen of hexadecimal data.

```

002A00  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  .....
002A10  10 11 12 13 14 15 16 17  00 01 02 03 04 05 06 07  .....
002A20  08 09 0A 0B 0C 0D 0E 0F  10 11 12 13 14 15 16 17  .....
002A30  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  .....
002A40  10 11 12 13 14 15 16 17  00 01 02 03 04 05 06 07  .....
002A50  08 09 0A 0B 0C 0D 0E 0F  10 11 12 13 14 15 16 17  .....
002A60  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  .....
002A70  10 11 12 13 14 15 16 17  00 01 02 03 04 05 06 07  .....
002A80  08 09 0A 0B 0C 0D 0E 0F  10 11 12 13 14 15 16 17  .....
002A90  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  .....
002AA0  10 11 12 13 14 15 16 17  00 01 02 03 04 05 06 07  .....
002AB0  08 09 0A 0B 0C 0D 0E 0F  10 11 12 13 14 15 16 17  .....
002AC0  18 00 00 00 00 00 00 00  00 00 00 00 00 00 00  .....
002AD0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00  .....
002AE0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00  .....
002AF0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00  .....

```

MVME117 1.x > MD 2A00 18      Display memory with a <count> of \$18.

```

002A00  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  .....
002A10  10 11 12 13 14 15 16 17  00 01 02 03 04 05 06 07  .....

```

MVME117 1.x > (CR)      Enter a carriage return to display the next <count> bytes (starting where the last request ended, even if in the middle of the line).

```

002A18  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  .....
002A28  10 11 12 13 14 15 16 17  00 01 02 03 04 05 06 07  .....

```

MVME117 1.x > (CR)

```

002A30  00 01 02 03 04 05 06 07  08 09 0A 0B 0C 0D 0E 0F  .....
002A40  10 11 12 13 14 15 16 17  00 01 02 03 04 05 06 07  .....

```

MVME117 1.x &gt;

**4.2.26 Memory Modify (MM)**MM  
M`M[M] <address>[;<options>]`

The function of the Memory Modify (M or MM) command is to change data in memory. An address and options are specified on the initial command line.

For convenient viewing and changing of object data, four variations of data updating capability are offered. These are enhanced by five options: the data size options, word and longword (the default size is byte); odd or even address access options (byte-size only); and a nonverification option for write-only operations. Action provided by an option specified on the initial command line is utilized in all four data updating submodes and remains in force until the M command is exited.

**4**

The five memory change mode options are:

- `;W` Set size to word (i.e., two bytes).
- `;L` Set size to longword (i.e., four bytes).
- `;O` Set size to byte; access only odd addresses.
- `;V` Set size to byte; access only even addresses.
- `;N` No verification. Do not read data after updating. If used, `;N` must be preceded by one of the above options (the semicolon (;) is required between multiple options).

When the memory change mode is entered on execution of the initial command line, object data in the specified locations is displayed in hexadecimal format, and the M command prompt (?) is presented at the right of the data. The data can then be changed using any of the subcommands described below. If desired, the action of the subcommand can be obtained without entering new data. For example, the contents of the preceding location(s) can be viewed by typing "`^(CR)`" alone after the ? prompt, or the M command can be exited by typing "`.(CR)`".

- `[<data>](CR)` Update location and sequence forward.
- `[<data>]^(CR)` Update location and sequence backward.
- `[<data>]=(CR)` Update location and reopen same location.
- `[<data>].(CR)` Update location and terminate.

MM  
M

### Disassemble/Assemble Mode (the ;DI option)

On execution of an initial M command line with the ;DI option selected, the disassemble/assemble mode is entered. Starting from the specified location, data is disassembled into a source instruction line, and both object data (in hexadecimal) and the source line are displayed. The M command prompt (?) is displayed to the right of the disassembled source line. If desired, a new source instruction may be typed and assembled to replace the existing instruction (the first character must be a space, which is recognized as the label field delimiter by the firmware monitor one-line assembler). Assembly is initiated by typing a carriage return. After assembly and updating, data in the following locations is disassembled and the next source line displayed. Note that the update and sequence backward (^ (CR)) and the update and reopen the same location (= (CR)) features are not available in the disassemble/assemble mode. Typing ". (CR)" while in this mode provides exit from the M command.

#### EXAMPLE

#### COMMENT

MVME117 1.x > M 10000;L

Memory modify location \$10000 a longword at a time.

010000 00000200 ? (CR)

No change to this longword.

010004 FFFFFFFF ? -5.

Change this longword to \$-5 (or \$FFFFFFFB) and stop.

MVME117 1.x > MM 10008;W;N

Modify memory location \$10008 a word at a time; do not read data after updating.

010008 ? 1111

Place a word of 1's into this location.

01000A ? 2222

Place a word of 2's into this location.

01000C ? 3333

Place a word of 3's into this location.

01000E ? 4444

Place a word of 4's into this location.

010010 ? ^

Back up one word.

01000E ? 5555

Place a word of 5's over the 4's in this location.

010010 ? 6666.

Place a word of 6's into this location and stop.

MVME117 1.x > M 20001;N;O

Place a byte into the odd locations only, without reading the data.

020001 ? 1=

Place a 1 into this memory location and remain at the same location. This technique is very useful for debugging I/O devices.

020001 ? 7=

Place a 7 at this memory location and remain at the same location.

020001 ? .

Exit the MM command.

MM  
M

EXAMPLE

MVME117 1.x > M 49528;DI

049528 48B800010406  
04952E 40F80406  
049532 48E7FFFE  
049536 4FF8095A

049536 4FF8095A  
049536 4FF8095A  
04953A 1600  
04953C 04444281

MVME117 1.x &gt;

COMMENT

MM starting at \$49528 using the disassembler.

MOVEM.W D0,\$0406 ? (CR)  
MOVE.W SR,\$0406 ? (CR)  
MOVEM.L D0-D7/A0-A6,-(A7) ? = (CR)  
LEA.L \$095A,A7 ? MOVE.B

X?(CR) (response to incomplete,  
LEA.L \$905A,A7 ? (CR) incorrect entry)  
MOVE.B D0,D3 ? (CR)  
SUB.W #17025,D4 ? ⋮
NOTE

Refer to Chapter 5 for more information  
about the assembler/disassembler.

#### 4.2.27 Memory Set (MS)

**MS**

MS <address> <data>

The Memory Set (MS) command changes the contents of memory. The data entered is placed at the location specified by <address>. If the data entered requires word alignment and <address> is not even, the byte at <address> is bypassed and the data is placed in the next even address.

Memory Set allows both hexadecimal and ASCII string data within the same line. The length of hexadecimal values can also vary. A space is used to delimit each field, and an apostrophe must be used to enclose each ASCII string.

Notice that lowercase is supported within the ASCII string. The firmware monitor's command and parameter parser automatically converts all lowercase input into uppercase. The only exceptions are the ASCII strings within apostrophes and the data entered while in transparent mode. This provides support for users wishing to use the terminal in lowercase. The commands and operands will work because they are converted to uppercase, and, where lowercase is specifically needed, it is supported (BS, MS, and TM).

The maximum number of bytes that can be entered with one MS <address> <data> command is limited to the size of the command line buffer, or 128 bytes. When the character in the last position of the first line is entered, an automatic CR/LF is sent to the display allowing the user to continue and still read the input characters entered.

##### EXAMPLE

##### COMMENT

MVME117 1.x > MD 2A00 30 Display memory at start before the MS command.

```
002A00  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
002A10  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
002A20  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
```

MVME117 1.x > MS 2A01 'MS Test' 41 61 42434445 20313233 'End of Data'

Enter ASCII string, two bytes, two longwords, and another string. (Notice lowercase input.)

MVME117 1.x > md 2a00 30 Now display results with the MD command. Notice command was entered in lowercase.

```
002A00  20 4D 53 20 54 65 73 74 20 41 61 42 43 44 45 20  MS Test AaBCDE
002A10  31 32 33 45 6E 64 20 6F 66 20 44 61 74 61 2E 20  123End of Data.
002A20  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
```

**MS**
**EXAMPLE**
**COMMENT**
**MVME117 1.x > BF 2A00 2B00 4161**

Fill memory with Aa so changes will stand out.

Physical Address=00002A00 00002B00

**MVME117 1.x > MD 2A70**

Display memory near end of what a full buffer can change.

002A70 41 61 41 61 41 61 41 61 41 61 41 61 41 61

**MVME117 1.x > MS 2A00 'This example shows that the MS command allows more than one line of data in the buffer at a time. Note the CR/LF sent' (CR)**

Enter long ASCII string.

**MVME117 1.x > md 2a00 80** Display a full 128 bytes.

002A00	54 68 69 73 20 65 78 61	6D 70 6C 65 20 73 68 6F	This example sho
002A10	77 73 20 74 68 61 74 20	74 68 65 20 4D 53 20 63	ws that the MS c
002A20	6F 6D 6D 61 6E 64 20 61	6C 6C 6F 77 73 20 6D 6F	ommand allows mo
002A30	72 65 20 74 68 61 6E 20	6F 6E 65 20 6C 69 6E 65	re than one line
002A40	20 6F 66 20 64 61 74 61	20 69 6E 20 74 68 65 20	of data in the
002A50	62 75 66 66 65 72 20 61	74 20 61 20 74 69 6D 65	buffer at a time
002A60	2E 20 4E 6F 74 65 20 74	68 65 20 43 52 2F 4C 46	. Note the CR/LF
002A70	20 73 65 6E 74 61 41 61	41 61 41 61 41 61 41 61	sentAaAaAaAaAa

**MVME117 1.x >**
**NOTE:** Carriage returns are not inserted into the data if screen wrap around occurs.

**NOTE:** If Printer Attach (PA) is being used, the print line may be exceeded, and data lost. (Prompt + MS + Address + 117 bytes of data can exceed the size of the printer buffer.)

## 4.2.28 Display Offsets (OF)

**OF**
**OF**

The OF command displays the offsets used to assist with relocatability and position-independent code.

When addresses are displayed (MD, DF, etc.), the appropriate offset register will be selected in the following manner:

1. If all registers are zero, the address is printed "as is", using no offset register.
2. If offset registers are nonzero, the register nearest the address (but still below), is subtracted. The register number and the remainder, now a displacement, is displayed.

Linked segments of code will each have a different load address or offset. For user convenience, seven general purpose offsets (.R0-.R6) are provided. Offset .R7 is always zero, which provides a convenient technique for entering an address without an offset. If no value is assigned to one of the general purpose offsets, it will have the default value of zero.

Unless another offset is entered, each command that expects an address parameter automatically adds offset R0 to the entered address -- that is, if R0 = 2000, the following commands are the same:

BR 10	(10 + 2000)	Offset R0 added by default.
BR 10+R0	(10 + 2000)	
BR 2010+R7	(2010 + 0)	R7 is always zero.

The physical address for each of these commands is 2010.

### EXAMPLE

### COMMENT

MVME117 1.x > <u>.R1 2000</u>	Set offset R1 to \$2000+.R0.
MVME117 1.x > <u>.R3 33000</u>	Set offset R3 to \$33000+.R0.
MVME117 1.x > <u>.R4 440000</u>	Set offset R4 to \$440000+.R0.
MVME117 1.x > <u>.R5 0</u>	Reset offset R5 to 0+.R0.
MVME117 1.x > <u>.R6 -1</u>	Set offset R6 to -1+.R0.
MVME117 1.x > <u>OF</u>	Display offsets.

R0=00000000 R1=00002000 R2=00000000 R3=00033000  
R4=00440000 R5=00000000 R6=FFFFFFFF R7=00000000

OF

**EXAMPLE**

MVME117 1.x > .R0 2200
**COMMENT**

Set offset R0 to 2200+.R0.

MVME117 1.x > MM 10  
000010+R0 61 ? .

Offset R0 is added to the address.

MVME117 1.x > MM 10+R7  
000010 00 ? .

R7 is always 0 and, when entered, overrides R0.

MVME117 1.x &gt;

**NOTE**

To set R0 to 0 after it has been set to a nonzero value, use the command ".R0 0+R7".  
The command ".R0 0" will not alter R0.

MVME117 1.x > OF Show current value of offset registers.  
R0-7 00000000 00002720 000027C2 00000000 00000000 00000000 00000000

MVME117 1.x > MD 9C+R1 5;DI Display memory showing end of one  
00009C+R1 B1C9 CMP.L A1,A0 module and beginning of another.

00009E+R1 66EE BNE.S \$0017AE

0000A0+R1 4E75 RTS

End of register 1's domain.

000000+R2 7404 MOVEQ.L #4,D2

Start of register 2's domain.

000002+R2 B1FC00F15000 CMP.L #15814656,A0

MVME117 1.x > .R1 0

Zero the offset registers.

MVME117 1.x > .R2 0

MVME117 1.x > MD 27BC 5;DI

Display the same area but with absolute addresses. (No offset registers.)

0027BC B1C9 CMP.L A1,A0

0027BE 66EE BNE.S \$0017AE

0027C0 4E75 RTS

0027C2 7404 MOVEQ.L #4,D2

0027C4 B1FC00F15000 CMP.L #15814656,A0

MVME117 1.x &gt;

**4.2.29 Printer Attach and Detach (PA and NOPA)**PA  
NOPAPA[<port number>]  
NOPA

The PA command allows the user to attach the line printer so that information sent to the console terminal will also be printed. Default is port 3 (MVME117 printer port).

Valid port numbers for this command are:

<u>PORT NUMBER</u>	<u>DESCRIPTION</u>
3	Specifies MVME117 Printer (connector on MVME708-1 Transition Module)
6	Specifies MVME050 Printer (connector on MVME701 Transition Module)
9	Specifies MVME410 Printer A (connector on MVME410)
A	Specifies MVME410 Printer B (connector on MVME410)

If the printer is deselected or not ready, the message PRINTER NOT READY will be sent to the console terminal. The firmware monitor will not wait until the printer is ready.

**NOTES**

- (1) Only one printer port can be attached at a time.
- (2) Some printers use pin 14 to enable an auto linefeed mode when pulled low. The MVME708-1 transition module connects pin 14 to ground as part of the standard printer interface. If one of the printers supporting this feature is used, the output will be double-spaced. Refer to the MVME117 Microcomputer User's Manual for a recommended solution.

The NOPA command allows the user to detach the line printer from the port previously attached. Output will then be displayed on the console terminal only; it will not be printed.

See also: MD, DU

PA  
NOPA

EXAMPLE
COMMENT

MVME117 1.x > PA

Assign default printer port (3).

MVME117 1.x > PA6

Assign printer port 6 (MVME050 printer).

MVME117 1.x > MD 2800 90

```
002800    FF FF 24 18 FF 7F 0C 00  FF 7F 30 04 FF FF 00 00  ..$......0.....
002810    FF 31 FF FF FF 01 FF FF  FF 26 FF FF 7F 22 FF FE  .1.....&..."~
```

:

Output is displayed on console terminal and printed at port 6.

MVME117 1.x > NOPA

Future output will be displayed on console terminal only.

MVME117 1.x >

**4.2.30 Port Format (PF)**

PF

PF[&lt;port number&gt;]

The Port Format (PF) command displays and optionally allows modification of serial port parameters. Depending on the port being specified any of the following characteristics may be specified:

- Baud rate
- Number of nulls (\$00) to be inserted following a carriage return
- Number of nulls (\$00) to be inserted following each character
- Number of bits per character
- Number of "Stop" bits
- Character parity

If the optional port number is not provided, each serial and printer port is displayed with its assigned port number as supported by the firmware monitor. Also, the Option Bytes address is displayed, and is used to optionally suppress the register display, and to contain the exit and trailing character for transparent mode. (Refer to TM command.)

If an invalid port number is specified with the PF command, the message: "Bad Port Number" is displayed and the MVME117 prompt returned.

If a valid port number is entered, then an interactive menu is presented to the user.

**NOTE**

Different ports may not allow every option, such as software programmable baud rate. This is controlled by a table within the firmware specifying the options that can be modified by the user.

All responses, both default and those entered by the user are saved in RAM until all of the parameters have been shown and optional replies entered. At this time, the message:

Ok (Y/N) ?

is displayed to the user. If correct, then "Y" should be entered followed by a carriage return. It is at this time that changes actually take place, and not as each parameter is entered.

PF

While responding to the prompts, there are four actions that can be taken following the question mark:

- ? (CR) -- Entering a carriage return indicates that the existing value being shown in the current prompt is acceptable; go to the next parameter.
- ? . -- Entering a period indicates that this menu session is to be terminated at this time, without any modifications being applied; control is returned to the MVME117 prompt.
- ? ^ -- Entering a caret symbol indicates that a previous parameter requires a change. The prompt will logically "back up" one parameter each time the symbol is entered until the first parameter is reached where it will remain until one of the other responses are received.
- ? <data> -- Entering the appropriate data requested (followed by a carriage return or ENTER).

Each of the parameters that are possible are now discussed along with any restrictions.

Baud rate is selected by entering a single digit at the following prompt:

1=19200  
2=9600  
3=2400  
4=1200  
5=300  
6=110

Baud Rate (1-6)=.....\$02 ? (Valid responses are 1 through 6.)

The amount of null (\$00) character padding to be inserted following each carriage return can be specified at the prompt:

C/R Nulls=.....\$00 ? (Valid responses are 0 through FE.)

The amount of null (\$00) character padding to be inserted following each character can be specified at the prompt:

Char Nulls=.....\$00 ? (Valid responses are 0 through FE.)

The number of bits per character are specified at the prompt:

Data Bits =.....\$08 ? (Valid responses are 5 through 8.)

PF

The number of stop bits per character are specified at the prompt:

Stop Bits =.....\$01 ? (Valid responses are 1 and 2.)

Parity is selected at the prompt:

Parity (E/O/N)=.....N ? (Valid responses are E, O, and N.)

### EXAMPLE

### COMMENT

MVME117 1.x > pf5

Request a change for MVME050 port 2.

1=19200  
2=9600  
3=2400  
4=1200  
5=300  
6=110

Baud Rate (1-5)=.....\$02 ? (CR)  
C/R Nulls=.....\$00 ? (CR)  
Char Nulls=.....\$00 ? (CR)  
Data Bits =.....\$08 ? 7  
Stop Bits =.....\$01 ? (CR)  
Parity (E/O/N)=.....N ? e

No change for baud rate.  
No change for carriage return nulls.  
No change for character nulls.  
Request 7 data bits per character.  
No change for stop bits.  
Request even parity.

Ok (Y/N) ? Y

Last chance. Go ahead.

PF complete

MVME117 1.x > pf7

Request a change for port 2 on the MVME400.

C/R Nulls=.....\$00 ? 1

Specify 1 null after each carriage return.

Char Nulls=.....\$00 ? (CR)

No change for character nulls.

Data Bits =.....\$08 ? 7

Specify 7 data bits per character

Stop Bits =.....\$01 ? (CR)

No change for stop bits.

Parity (E/O/N)=.....N ? e

Specify even parity

Ok (Y/N) ? Y

Last chance. Go ahead.

PF complete

NOTE: The baud rate for the MVME400 is specified with jumpers on the module.

MVME117 1.x >

PF

**EXAMPLE**

 MVME117 1.x > pf

Options @ 04E8

Ports:

MVME117	MVME050	MVME400	MVME410
1=Term1	4=Term1	7=Term2	9=PRT A
2=Term2	5=Term2	8=Term1	A=PRT B
3=PRT	6=PRT		

MVME117 1.x &gt;

**COMMENT**

Find the option bytes address and the port number assignments.

The following are examples both with and without register information within Trace and Breakpoint execution.

**EXAMPLE**

 MVME117 1.x > PF

Options @ 04E8

Ports:

MVME117	MVME050	MVME400	MVME410
1=Term1	4=Term1	7=Term2	9=PRT A
2=Term2	5=Term2	8=Term1	A=PRT B
3=PRT	6=PRT		

 MVME117 1.x > .PC 50000

 MVME117 1.x > T 3

Physical Address=00050000

PC=00050006 SR=2704=.S7.Z.. USP=00070000 SSP=00050000 VBR=00000000 SFC=2 DFC=1

DO-7 44306162 44316364 44326566 44336768 4434696A 44356B6C 44366D6E 44376F70

AO-7 00050100 41317374 41327576 41337778 4134797A 000401CE 000401E1 00050000

PC=050006 203C0000000A MOVE.L #10,D0

PC=0005000C SR=2700=.S7.... USP=00070000 SSP=00050000 VBR=00000000 SFC=2 DFC=1

DO-7 0000000A 44316364 44326566 44336768 4434696A 44356B6C 44366D6E 44376F70

AO-7 00050100 41317374 41327576 41337778 4134797A 000401CE 000401E1 00050000

PC=05000C 3100 MOVE.W D0,-(A0)

PC=0005000E SR=2704=.S7.... USP=00070000 SSP=00050000 VBR=00000000 SFC=2 DFC=1

DO-7 0000000A 44316364 44326566 44336768 4434696A 44356B6C 44366D6E 44376F70

AO-7 000500FE 41317374 41327576 41337778 4134797A 000401CE 000401E1 00050000

PC=05000E 51C8FFFC DBF.L D0,\$05000C

 MVME117 1.x :> M 04E8

0004E8 00 ? (CR)

0004E9 00 ? (CR)

0004EA 00 ? (CR)

 0004EB 00 ? 1.
**COMMENT**

Invoke PF command to display "options" location.

The fourth byte within the options RAM area is initialized to zero, allowing full register display for each instrumentation traced or each breakpoint reached.

Set PC for program execution.

Trace three instructions, with register display.

Now examine and modify the fourth byte within the options RAM to suppress the register display.

PF

**EXAMPLE**
**COMMENT**

MVME117 1.x > T\_6  
 Physical Address=0005000E  
 PC=05000C 3100  
 PC=05000E 51C8FFFC  
 PC=05000C 3100  
 PC=05000E 51C8FFFC  
 PC=05000C 3100  
 PC=05000E 51C8FFFC

Trace the next six instructions without displaying the registers.  
 MOVE.W D0,-(A0)  
 DBF.L D0,\$05000C  
 MOVE.W D0,-(A0)  
 DBF.L D0,\$05000C  
 MOVE.W D0,-(A0)  
 DBF.L D0,\$05000C

MVME117 1.x >: BR 5000C

Set a breakpoint within the execution path to illustrate reaching a breakpoint with the register display suppressed.

Breakpoints  
 05000C 05000C  
 MVME117 1.x > .PC 50000  
 MVME117 1.x > GO  
 Physical Address=00050000

Set the PC back to the start of the routine. Begin execution.

At Breakpoint

NOTE: Only the PC and the disassembled instruction are displayed.

PC=05000C 3100  
 MVME117 1.x > GO  
 Physical Address=0005000C

MOVE.W D0,-(A0)  
 Continue execution.

At Breakpoint  
 PC=05000C 3100

Breakpoint reached.  
 MOVE.W D0,-(A0)

Even with the register display suppressed the DF command can still be used to examine the registers when needed.

MVME117 1.x > DF  
 PC=0005000C SR=2700=.S7.... USP=00070000 SSP=00050000 VBR=00000000 SFC=2 DFC=1  
 D0-7 00000009 44316364 44326566 44336768 4434696A 44356B6C 44366D6E 44376F70  
 A0-7 000500FE 41317374 41327576 41337778 4134797A 000401CE 000401E1 00050000  
 PC=05000C 3100 MOVE.W D0,-(A0)  
 MVME117 1.x >

**4**

#### 4.2.31 Power-Up Test Enable/Disable (PT and NOPT)

PT  
NOPTPT  
NOPT

The PT and NOPT commands allow the user to select whether or not the MVME117 is to execute the Power-Up Test during the power-up sequence. PT enables the test; NOPT disables it. Refer to paragraph 2.3.

Note that the Power-Up Test is only invoked at power-up and not each time the RESET switch is pressed. If a confidence check is desired after power-up, the Self Test (ST) command can be used. This initiates the applicable power-up tests and a more complete set of diagnostics.

For more specific information on the Power-Up Test, refer to paragraph 2.6.2. For more specific information on the ST command, refer to paragraph 4.2.34.

CMOS RAM which is battery backed-up is used to contain the enable/disable status. The actual address used is as follows:

\$F43FC9 = Power-Up Test switch    ASCII 'Y' = Power-Up Test Enabled (default).  
   ASCII 'N' = Power-Up Test Disabled.

#### NOTES

- (1) Only odd addresses are provided in CMOS RAM. An \$F is displayed for each even address when Memory Display (MD) command is used.
- (2) When using a 2K x 8 CMOS RAM, such as the one installed at the factory, its addresses are echoed at \$F40Fxx, \$F41Fxx, \$F42Fxx, and \$F43Fxx. However, this is not true when an 8K x 8 CMOS RAM is installed.

### 4.3.32 SCSI/SASI Bus Reset or SCSI Controller Reset (RESET)

**RESET**

RESET [<SCSI-level>]

The RESET command is used to issue a general SCSI/SASI bus reset, or to issue a specific SCSI controller reset if the controller number (or SCSI-level) is provided on the command line.

For those controllers that are said to be SCSI-compatible but do not support the full SCSI protocol (specifically, they do not support a message out phase which must be available to issue a specific controller reset), a general SCSI bus reset is provided. By entering the RESET command without a controller number, the user is requesting the general bus reset.

For those controllers that truly provide the complete SCSI protocol, the controller number (SCSI ID-level) can be entered on the command line to request that only that controller be reset.

The attach packet save areas in CMOS RAM are invalidated (\$FFFFFFFF moved to the controller number or ID-level and status), forcing any new packets to be built from scratch. (Refer to Appendix E for details of SCSI attach packet format.)

#### EXAMPLE

#### COMMENT

MVME117 1.x > RESET 3

SCSI Disk Error: 200B0000

PC=00000000 SR=2704=.S7..Z..USP=00000000  
D0-7 00000118 00000000 00000007 00000000  
A0-7 00F0B49A 00F01F4A 00000D22 00000000  
PC=000000 00F0 DC.W \$00F0

MVME117 1.x > RESET

SCSI "Bus Reset" complete

MVME117 1.x >

In the first example, a DTC 520-B controller is connected to the SCSI bus as ID-level 03.

A specific controller reset is issued with a \$0B returned indicating a command error from the SCSI firmware (this controller does not support the message out phase).

SSP=00001800 VBR=00000000 SFC=0 DFC=0  
00002028 0000002C 00000014 0000200B  
00F01834 000006C0 00000D22 000001800

A general SCSI bus reset is requested.

Normal completion is indicated.

MVME117 1.x > RESET 3

SCSI "Controller Reset" complete

MVME117 1.x >

In this example, a Saber-AP controller is connected to the SCSI bus as ID-level 03. Normal completion of the specific controller reset is indicated.

#### 4.2.33 Set Date and Time of Day (SET)

**SET**
**SET**

The SET command is interactive and begins with the user entering the ASCII string "SET" followed by a carriage return. At this time, a prompt asking for MM/DD/YY is displayed. The user should respond as indicated. Note that an incorrect entry may be corrected by backspacing or deleting the entire line as long as the carriage return has not been entered.

After the initial prompt and entry, another prompt is presented asking for HH:MM:SS. Again, corrections can be applied in a similar manner. When the correct time matches the data entered, the user should press the carriage return to establish the current value in the time of day clock.

Note that the ASCII data entered for both the date and time is written into CMOS RAM where a simple Memory Display (MD) command will show when the onboard time-of-day clock was last set. Note, however, that CMOS RAM only supports the odd addresses, with \$F's being displayed in unpopulated RAM locations. NOTES on Autoboot (AB) command, paragraph 4.2.2, have additional discussion of address differences in 2K x 8 CMOS RAM and in 8K x 8 CMOS RAM.

After this information is used to initialize the clock and document when the SET command was entered, the TIME command is entered by default. This presents the ASCII date and time on the terminal. Be advised that the numbers are updated as they change and if Printer Attach (PA) is in effect, one line will be printed for each time the clock changes. This will also be seen if a hard copy terminal is used as a debug terminal.

To terminate the TIME command, press the BREAK key. Any interrupts that might have been pending will be cleared before the MVME117 prompt is displayed. The TIME command is the only area within the firmware that recognizes and uses interrupts.

##### EXAMPLE

```
MVME117 1.x > SET
Enter date as MM/DD/YY
10/03/85
Enter time as HH:MM:SS (24 HOUR CLOCK)
08:22:50
10/03/85 08:22:50
```

Break

```
MVME117 1.x > MD F43FC0 30
```

```
F43FC0 FF 00 FF 00 FF 00 FF 80 FF 59 FF 4E FF 32 FF 31 .....Y.N.2.1
F43FDO FF 31 FF 30 FF 2F FF 30 FF 33 FF 2F FF 38 FF 35 .1.0./0.3./8.5
F43FE0 FF 30 FF 38 FF 3A FF 32 FF 32 FF 3A FF 35 FF 30 .0.8.:2.2.:5.0
MVME117 1.x >
```

##### COMMENTS

Invoke SET command.

User enters date, followed by a (CR).

User enters a value for time and when it is 8:22:50 presses (CR).

Press BREAK key to exit. Terminal responds with "Break".

Display CMOS RAM where firmware monitor saves its variables.

#### 4.2.34 Self-Test (ST)

ST

ST [&lt;device&gt;] [,&lt;controller&gt;]

where:

<device> is a single hexadecimal digit, 0 through 7, specifying the disk to be read.

<controller> is a single hexadecimal digit, 0 through 8, specifying the controller through which the disk is connected.

The ST command performs a more extensive diagnostic test of the VME system than the power-up test. When the test is entered, the FAIL LED is turned on. If the test passes, a pass message is displayed on the current terminal and the FAIL LED is turned off. If the test fails, a failure message is displayed at the current terminal if possible. The FAIL LED is left on and control is returned to the reset entry point of the debug monitor. Depressing RESET and performing a valid warm start (refer to paragraph 2.2.1) always turns off the FAIL LED.

The following tests are performed:

a. MC68010 MPU Test

Registers, arithmetic logic unit, instructions, and exception processing are tested using the MC68010.

b. ROM Test

16-bit checksum is calculated using the EXCLUSIVE-OR of all ROM data.

c. RAM Test - Parity Disabled

All CPU RAM above address \$FFF is tested using the walking bit (0 through a field of 1's and 1 through a field of 0's), and pattern (\$00000000 then \$FFFFFFFF) tests. Parity checking is disabled during these tests. The size of onboard RAM is verified with the module configuration number.

d. Parity Logic Test

Testing of the parity circuitry is performed by the following tests:

- . Verification that incorrect parity can be written and detected in upper and lower bytes.
- . Check that incorrect parity causes a bus error.
- . Check that Parity Error bit is set in CPU Control Register.
- . Verification that parity checking can be disabled.

**ST**
**e. Parity RAM Test**

With parity enabled, data \$0001 is written to all addresses above \$FFF and then verified. Data \$0000 is then written and verified.

**f. RAM Test - Parity Enabled**

With parity enabled, all CPU RAM above address \$FFF is tested using the walking bit (0 through a field of 1's and 1 through a field of 0's), and pattern (\$00000000 then \$FFFFFFFF) tests.

**g. CMOS RAM Test**

CMOS memory is sized for 2K or 8K. When the size has been determined, it is displayed after the test message. A non-destructive memory test is then performed on the size found.

**h. Serial Port Test**

Because one of several serial ports is used as the system console, this test runs the internal loopback test on the two MVME117 ports. The messages "THIS IS A TEST OF PORT 1" and "THIS IS A TEST OF PORT 2" are transmitted to the appropriate ports and verified. Due to the characteristics of the loopback mode, these characters are also transmitted to any device connected to the MVME117 ports.

If this test fails, the user receives an error code displayed thus:

```
Serial Port Test  FAILED
Error Code  $xxxx
```

The error code may be deciphered using the following description:

```

15-----0
Error Code = xxxx xxxx xxxx xxl1 Port #1 Transmit Error
            = xxxx xxxx xxxx xxlx Port #1 External/Status Change
7--0      = xxxx xxxx xxxx xlx1 Port #1 Receive Error
            = xxxx xxxx xxxx lxxx Port #1 Special Rx Condition
            = xxxx xxxx xxx1 xxxx Port #1 Transmit Time-out
            = xxxx xxxx xxlx xxxx Port #1 Receive Time-out
            = xxxx xxxx xlx1 xxxx Not Used
            = xxxx xxxx lxxx xxxx Not Used

Error Code = xxxx xxx1 xxxx xxxx Port #2 Transmit Error
            = xxxx xlx1 xxxx xxxx Port #2 External/Status Change
15--8     = xxxx xlx1 xxxx xxxx Port #2 Receive Error
            = xxxx lxxx xxxx xxxx Port #2 Special Rx Condition
            = xxx1 xxxx xxxx xxxx Port #2 Transmit Time-out
            = xlx1 xxxx xxxx xxxx Port #2 Receive Time-out
            = xlx1 xxxx xxxx xxxx Not Used
            = lxxx xxxx xxxx xxxx Not Used
```

ST

## i. PTM Test

Timers 1, 2, and 3 are set for maximum count and allowed to time out using internal clocks, with outputs disabled.

Timers 1, 2, and 3 are set to interrupt when they time out, using internal clocks. The outputs of timers 1 and 2 are disabled.

Timer 3 output is enabled to provide external clocks to timer 1. The output of timer 1 is disabled and both timers are allowed to time out.

## j. Real-Time Clock - TOD Test

The MM58274 is checked for rollover in time from

99:12:31:23:59:59: day 7 to  
00:01:01:00:00:00 day 1

## k. MC68881 Test

A detection of the MC68881 chip is verified with the module configuration number. If the MC68881 chip is detected, then several data conversion tests and a multiplication test are performed.

## l. SCSI Test

A detection of the SCSI chip is verified with the module configuration number. The detection is performed by setting the SCSI chip in the Test Mode and verifying its registers. The chip is then reset via the MCR register (bit 6), and its registers are read and verified. No other tests are performed at this time. Refer to "Disk/Controller Test" for SCSI offboard testing.

## m. Disk/Controller Test

Using the controller and drive information supplied by the ST command, a read of sector 0 is performed. This test is only performed if valid device and controller numbers are entered on the "ST" command line. If no parameters are entered, the default is to bypass this test.

With the media in the VERSAdos 256-byte block format, the 64-byte diagnostic pattern is verified as well as the MOTOROLA or EXORMACS ASCII character string.

ST

## n. ABORT Switch Test

The operator presses the ABORT switch on the front panel to verify the switch. A single carriage return bypasses this test.

## o. LED Test

Tests control register fail bit for correct operation. Visual verification of LED states by operator is performed.

## p. Front Panel Switch Test

This tests the eight parts of the software readable switch located on the front panel, by having the operator verify the switch positions.

If any tests fail, appropriate messages are displayed.

**NOTE**

The ST command is destructive. The debugger RAM is cleared and the firmware monitor returns to its initialized state.

**ST**
**EXAMPLE**
**MVME117 1.x > ST 0,2**

```

68010 MPU Test  PASSED
ROM Test  PASSED
RAM Test - Parity Disabled (Size = 512K)  PASSED
Parity Logic Test  PASSED
Parity RAM Test  PASSED
RAM Test - Parity Enabled  PASSED
CMOS Memory Test (Size = 2K)  PASSED
Serial Port Test  This is a test of Port #1  PASSED
PTM Test  PASSED
Real Time Clock Test  PASSED
68881 Test  PASSED
SCSI Test  PASSED
Disk/Controller Test  PASSED
Abort Button Test  PASSED
LED Test IS THE FAIL LED OFF? Y/N <CR> Y
LED Test IS THE FAIL LED ON? Y/N <CR> Y
LED Test  PASSED
Front Panel Switch Test
    Is 1,3,5,7 On & 2,4,6,8 Off Y/N <CR> or only <CR> to read again Y
Front Panel Switch Test  PASSED

```

**SELF-TEST PASSED**
**Size and Initialize RAM (Y/N) ? Y**
**COLD Start**

```

    End Onboard RAM = $01FFFF
    Start Offboard RAM = $180000
    End Offboard RAM = $1BFFFF

```

**MVME117 1.x >**
**COMMENT**

Invoke the ST; with a check of sector 0 of drive 0, controller 2. If the numbers are wrong, the user gets a Syntax Error or a SCSI Disk Error at the time of the Disk/Controller Test.

Must answer both questions Y to pass.

The self-test detected no failures.

Let the firmware monitor search for the end of the contiguous RAM, and initialize the RAM that it finds.

The firmware monitor displays the cold start message.

The firmware monitor is ready to accept the next command.

#### 4.2.35 Terminal Attach (TA)

TA

TA[&lt;port number&gt;]

The TA command allows the user to execute the debug monitor from a serial port other than the MVME117 onboard debug port. The TA command prevents a port conflict in a multiprocessor configuration.

The optional port number in the command line directs the debug monitor to accept its input from, and display its output to the port specified.

Valid port numbers for this command are:

<u>PORT NUMBER</u>	<u>DESCRIPTION</u>
none	Default MVME117 port 1 (connector on MVME117)
1	Specifies MVME117 port 1 (connector on MVME117)
2	Specifies MVME117 port 2 (connector on MVME708-1 Transition Module)
4	Specifies MVME050 terminal 1 (connector on MVME701 Transition Module)
5	Specifies MVME050 terminal 2 (connector on MVME701 Transition Module)
7	Specifies MVME400 terminal 2 (connector on MVME400)
8	Specifies MVME400 terminal 1 (connector on MVME400)

To return control to the debug port on the MVME117, a TA command with no following port number can be issued. The VI command and any cold start will also return control to the MVME117 onboard port (refer to the VI command).

After control has been established at the new port, all commands which direct input from or output to ports remain valid (including PA).

**4.2.36 Display Current Date and Time of Day (TIME)****TIME****TIME**

This command presents the date and time in ASCII characters from the console. Be advised that the numbers are updated as they change, and if Printer Attach (PA) is in effect, one line is printed each time the clock changes. This is also seen if a hard copy terminal is used as a debug terminal.

To terminate the TIME command, press the BREAK key. Any interrupts that might have been pending will be cleared before the MVME117 prompt is displayed. The TIME command is the only area within the firmware that recognizes and uses interrupts.

To initialize the time-of-day clock, refer to the SET command, paragraph 4.2.33.

**EXAMPLE**

MVME117 1.x > TIME

Press "Break" to exit

06/12/85 07:40:28

Break

MVME117 1.x >

**COMMENT**

Invoke the TIME command requesting the display of the date and time of day.

Press BREAK key to exit. Terminal responds with "Break".

### 4.2.37 Transparent Mode (TM)

**TM**

TM[<port number>] [<exit character>[<trailing character>]]

The TM command, together with the onboard serial ports (an MVME400 dual RS-232C serial port I/O module, or a serial port on an MVME050 system controller module), provides terminal support in a dumb terminal fashion until the exit character is received.

The <exit character> is entered right after the port number, though an optional space is permitted. Note that CTRL-X, CTRL-D, CTRL-H, CTRL-J, and CTRL-M cannot be specified as exit or trailing characters from the TM command. These characters provide control for the terminal (e.g., line delete, redisplay the line, backspace) and will not be passed to the TM command from the terminal handler. All other characters, both CTRL and non-CTRL, can be entered. (Default <exit character> is CTRL-A.)

The <trailing character> is transmitted to the host port upon receipt of the <exit character>. With systems using VERSAdos, the standard trailing character is a CTRL-X which cancels anything that might have progressed to an input buffer. By default, CTRL-X is specified as the trailing character so there is no problem using it (though if another trailing character were selected, CTRL-X could not be entered to reinstate it).

There is an alternate way of setting the exit and trailing characters. Since they are stored in RAM, any user knowing the location of these characters could use a Memory Modify (MM) command to alter those values. By issuing a Port Format (PF) command, the address of the MVME117 debug monitor option bytes is displayed in the display.

Trailing character is located at offset \$4  
Exit character is located at offset \$5

In systems where no data at all must be sent upon exit of transparent mode, the trailing character can be changed to \$00 using MM within the option bytes. With the trailing character as NUL, there is no data sent at exit time.

Valid port numbers for this command are:

<u>PORT NUMBER</u>	<u>DESCRIPTION</u>
none	Default MVME117 port 1 (connector on MVME117)
1	Specifies MVME117 port 1 (connector on MVME117)
2	Specifies MVME117 port 2 (connector on MVME708-1 Transition Module)
4	Specifies MVME050 terminal 1 (connector on MVME701 Transition Module)
5	Specifies MVME050 terminal 2 (connector on MVME701 Transition Module)
7	Specifies MVME400 terminal 2 (connector on MVME400)
8	Specifies MVME400 terminal 1 (connector on MVME400)

The first time the TM command is used, if no port number is specified, the default is port 1. For subsequent uses of TM, if no port number is specified, the default is the port used the last time.

**TM**
**EXAMPLE**

MVME117 1.x > TM4

\*TRANSPARENT\* EXIT=\$01 = CTL A

MVME117 1.x > TM4 (CTRL-W)

\*TRANSPARENT\* EXIT=\$17 = CTL W

MVME117 1.x > PF

Options @ 04E8

Ports:

MVME117	MVME050	MVME400	MVME410
1=Term1	4=Term1	7=Term2	9=PRT A
2=Term2	5=Term2	8=Term1	A=PRT B
3=PRT	6=PRT		

MVME117 1.x > M 04E8;W

0004E8	0000 ? (CR)
0004EA	0000 ? (CR)
0004EC	1817 ? <u>001C.</u>

MVME117 1.x > TM4

\*TRANSPARENT\* EXIT=\$1C = CTL \

**COMMENT**

Go transparent through serial port 1 on the MVME050 and request default exit and trailing character by entering TM4.

Debug monitor responds with both hex and CTRL display of exit character.

Specify CTRL-W as exit character with TM (hold down the CTRL key and press the W key). The firmware monitor responds with hex 17 and CTRL-W, verifying what was entered.

Use PF command to show where the firmware monitor option bytes are located within RAM.

Use MM command to display the trailing character (\$18) and the current exit character (\$17). Then change the trailing character to prevent transmission of any data upon exit (refer to the discussion of null trailing characters) and change exit character to a CTRL-\ (control backslash).

Enter a TM command without any operands (except the port number). Previous values will be used, as changed by MM.

**4**

TM

EXAMPLEMVME117 1.x > M 04EC;W0004EC 001C ? 1801.MVME117 1.x > TM4

\*TRANSPARENT\* EXIT =\$01 = CTL A

MVME117 1.x &gt;

COMMENT

Change trailing character back to CTRL-X and exit character to CTRL-A.

Enter TM4 and the CTRL-A exit character is verified.

#### 4.2.38 Trace (TR)

TR  
T

T[R] [&lt;count&gt;]

The Trace (T or TR) command executes instructions one at a time, beginning at the location pointed to by the program counter. After execution of each instruction, the MC68010 registers are displayed.

When the trace mode is entered, the prompt includes a colon (i.e., MVME117 1.x :>). While in this mode, typing only a carriage return will cause one instruction to be traced.

Breakpoints and breakpoint counts are in effect during trace.

Trace cannot be used to step through interrupts or exceptions (e.g., TRAP).

<u>COMMAND FORMAT</u>	<u>DESCRIPTION</u>
MVME117 1.x > I	Trace one instruction.
MVME117 1.x :> <u>TR &lt;count&gt;</u>	Trace <count> instructions.
MVME117 1.x :> ( <u>CR</u> )	Carriage return (CR) executes next instruction.
MVME117 1.x :> <u>MD 1000</u>	Typing the next command exits trace mode.
MVME117 1.x >	

#### NOTE

If the program counter contains an address that falls between the starting and ending addresses of the firmware monitor program, the warning message .PC within "DEBUGGER" will be returned. Processing will continue with unexpected results if stack pointers and/or registers are not handled properly.

See also: DF, GO, GT, PF (Options @ xxxx), TT

TR  
T

EXAMPLE:

MVME117 1.x > .PC 2000

MVME117 1.x > IR

Physical Address=00002000

PC=00002002 SR=2700=.S7.....USP=FFFFFFFF SSP=00000800 VBR=00000000 SFC=2 DFC=2

D0-7 00304E71 00002000 C05E2000 00000000 1796AF30 00000020 00000000 00000000

A0-7 00F021CA 00000000 00002000 00000458 00000410 00000551 00000551 00000800

PC=002002

MVME117 1.x :> (CR)

Physical Address=00002002

PC=00002004

D0-7 00304E71 00002000 C05E2000 00000000 1796AF30 00000020 00000000 00000000

A0-7 00F021CA 00000000 00002000 00000458 00000410 00000551 00000551 00000800

PC=002004

MVME117 1.x :> T 2

Physical Address=00002004

PC=00002006 SR=2700=.S7.....USP=FFFFFFFF SSP=00000800 VBR=00000000 SFC=2 DFC=2

D0-7 00304E71 00002000 C05E2000 00000000 1796AF30 00000020 00000000 00000000

A0-7 00F021CA 00000000 00002000 00000458 00000410 00000551 00000551 00000800

PC=002006

PC=00002008 SR=2700=.S7.....USP=FFFFFFFF SSP=00000800 VBR=00000000 SFC=2 DFC=2

D0-7 00304E71 00002000 C05E2000 00000000 1796AF30 00000020 00000000 00000000

A0-7 00F021CA 00000000 00002000 00000458 00000410 00000551 00000551 00000800

PC=002008

MVME117 1.x :&gt;

#### 4.2.39 Trace to Temporary Breakpoint (TT)

TT

TT &lt;breakpoint address&gt;

The TT command performs the following:

- a. Sets a temporary breakpoint at the address specified.
- b. Starts program execution in the trace mode.
- c. Traces until any breakpoint with a zero count is encountered.
- d. Resets the temporary breakpoint.

The temporary breakpoint is not displayed by the BR command.

When the trace mode is entered, the prompt includes a colon (i.e., MVME117 1.x :>). While in this mode, typing only a carriage return will cause one instruction to be traced.

See also: DF, GO, GT, TR

#### EXAMPLE

MVME117 1.x > .PC 2000

MVME117 1.x > TT 2006

Physical Address=00002006

Physical Address=00002000

PC=00002002 SR=2700=.S7.....USP=FFFFFFF SSP=00000800 VBR=00000000 SFC=2 DFC=2

D0-7 00304E71 00002000 C05E2000 00000000 1796AF30 00000020 00000000 00000000

A0-7 00F021CA 00000000 00002000 00000458 00000410 00000551 00000551 00000800

PC=002002

PC=00002004 SR=2700=.S7.....USP=FFFFFFF SSP=00000800 VBR=00000000 SFC=2 DFC=2

D0-7 00304E71 00002000 C05E2000 00000000 1796AF30 00000020 00000000 00000000

A0-7 00F021CA 00000000 00002000 00000458 00000410 00000551 00000551 00000800

PC=002004

At Breakpoint

PC=00002006 SR=2700=.S7.....USP=FFFFFFF SSP=00000800 VBR=00000000 SFC=2 DFC=2

D0-7 00304E71 00002000 C05E2000 00000000 1796AF30 00000020 00000000 00000000

A0-7 00F021CA 00000000 00002000 00000458 00000410 00000551 00000551 00000800

PC=002006

MVME117 1.x :>

#### 4.2.40 Verify (S-Records) (VE)

**VE**

VE[<port number>] [;<options>] =<text>

The VE command prepares the MVME117 to receive S-records from the designated <port number> and then transmits the <text> following the = sign to the system connected to the <port number> indicated. As the S-records are received, the bytes are compared one by one with the contents of memory. If all bytes are correct, the firmware monitor prompt is returned. However, if any data does not compare, a message in the following format is displayed:

```
S1LLaaaa.-.-.-.-.-DD.-.-.-.-.-
                        or
S2LLaaaaaa.-.-.-.-.-DD.-.-.-.-.-
```

where:

S1 or S2	is the S-record type (note size of address).
LL	is the length of the data contained within the checksum.
aaaa or aaaaaa	is the address that this S-record verifies.
.-.-.-.-	are characters that verify correctly.
DD	represents the contents of the S-record where data was found to be different.

Refer to Appendix D for a discussion of S-record content.

The following options are supported:

;-C	Ignore checksum validation on each S-record while loading.
;X	Echo the S-records read to the console terminal.

Default source is the current terminal attached. Specifying VE<port number> allows the input to be received from other ports.

**VE**

Valid port numbers for this command are:

<u>PORT NUMBER</u>	<u>DESCRIPTION</u>
none	Default MVME117 port 1 (connector on MVME117)
1	Specifies MVME117 port 1 (connector on MVME117)
2	Specifies MVME117 port 2 (connector on MVME708-1 Transition Module)
4	Specifies MVME050 terminal 1 (connector on MVME701 Transition Module)
5	Specifies MVME050 terminal 2 (connector on MVME701 Transition Module)
7	Specifies MVME400 terminal 2 (connector on MVME400)
8	Specifies MVME400 terminal 1 (connector on MVME400)

**EXAMPLE**
**COMMENT**

MVME117 1.x > BF 2D00 2E00 4161  
Physical Address=00002D00 00002E00

Initialize RAM to known pattern.

MVME117 1.x > MD 2D00  
002D00 41 61 41 61 41 61 41 61 41 61 41 61 41 61 AaAaAaAaAaAaAaAa

Display memory.

MVME117 1.x > L02=DU 2D00 2D80  
2D00 2D80

Enter LO command which contains a Dump (DU) command for an Educational Computer Board (ECB).

MVME117 1.x > MD 2D00 8F Display RAM to see if data was transferred.

```

002D00 00 01 02 03 04 05 FF FF 08 09 0A 0B 0C 0D 0E 0F .....
002D10 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F .....
002D20 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F !"#$%&'()*+,-./
002D30 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 0123456789:;<=>?
002D40 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F @ABCDEFGHIJKLMNO
002D50 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F PQRSTUVWXYZ[\]^_
002D60 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 'abcdefghijklmnopqrstuvwxyz{|}~.
002D70 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F pqrstuvwxyz{|}~.
002D80 80 61 41 61 41 61 41 61 41 61 41 61 41 61 .aAaAaAaAaAaAaAa

```

MVME117 1.x > M 2D78  
002D78 78 ? 88.

Alter two bytes with MM to see if VE can detect incorrect bytes.

MVME117 1.x > M 2D1A  
002D1A 1A ? 0A.

**VE**
**EXAMPLE**
**COMMENT**

MVME117 1.x > VE2=DU 2D00 2D80  
 DU 2D00 2D80  
 S1132D10.-.-.-.-.-1A.-.-.-.-.-  
 S1132D70.-.-.-.-.-78.-.-.-.-.-

Enter the VE command requesting the same S-records be transmitted.

Two errors are detected.

MVME117 1.x > M 2D1A  
 002D1A 0A ? 1A

Use MM to change the two bytes back to the correct values.

MVME117 1.x > M 2D78  
 002D78 88 ? 78.

MVME117 1.x > VE2=DU 2D00 2D80

Invoke the VE command again; no errors are detected this time.

MVME117 1.x >

**4.2.41 Vector Initialize (VI)**

VI

## VI

The VI command provides a way of initializing all system and user vectors.

A user may modify various vectors for his own use but then decide to set them back to the way the firmware monitor initialized them. However, because of the warm start feature on the MVME117 1.x firmware monitor, RESET will not initialize the vectors, unless one of the vectors vital to the firmware monitor's operation has been altered. (Refer to paragraph 2.2.1, RESET.) The VI command forces the initialization of the vectors, and goes through a cold start sequence.

At RESET, the firmware monitor inspects the vectors that are required for operation of the monitor to see if they are intact. If so, the message:

WARM Start

will then be displayed on the terminal. In addition, the bus error vector is initialized to the firmware monitor value. No other vectors will then be examined or modified. If, however, any of these vectors is not correct, all vectors will then be initialized, followed by the RAM sizing question.

EXAMPLECOMMENT

MVME117 1.x > VI

Force the vectors to be initialized, and reset the firmware monitor to its cold start state.

Size and Initialize RAM (Y/N) ? Y

Let the firmware monitor search for the end of the contiguous RAM, and initialize the RAM that it finds.

COLD Start

The firmware monitor displays the cold start message.

End Onboard RAM = \$01FFFF  
Start Offboard RAM = \$180000  
End Offboard RAM = \$1BFFFF

MVME117 1.x >

The firmware monitor is ready to accept a command.

### 4.3 COMMAND SUMMARY

The commands and options available to MVME117 1.x users are summarized in Table 4-2.

TABLE 4-2. Firmware Monitor Command and Option Summary

COMMAND	DESCRIPTION
[NO]AB [<device>,<controller>]	Enable/disable Autoboot.
BD [<device>] [,<controller>]	Boot dump.
BF <address1> <address2> <pattern>	Block fill.
BH [<device>] [,<controller>]	Boot and halt.
BI <address1> <address2>	Block initialize.
BM <address1> <address2> <address3>	Block move.
BO [<device>] [,<controller>] [<string>]	Boot operating system.
[NO]BR [<address> [;<count>]]	Set and remove breakpoints.
BS <address1> <address2> '<literal string>' <address1> <address2> <data> [<mask>] [;<option>]	Block search; options ;B ;W ;L ;-B ;-W ;-L.
BT <address1> <address2>	Block test.
CMOS	Display CMOS RAM variables.
CS [<address1>] [<address2>]	Checksum.
DC <expression>	Data conversion/evaluation.
DF	Display formatted registers.
DU[<port number>] <address> <address2> [<test>]	Dump memory (S-records).
GD [<address>]	Go direct.
G[0] [<address>]	Install breakpoints and go.
GT <temporary breakpoint address>	Go until address.
HE	Display commands/registers.
IOC	Issue RWIN command.

**TABLE 4-2. Firmware Monitor Command and Option Summary (cont'd)**

COMMAND	DESCRIPTION
IOP	Issue physical read/write.
IOT [<device>] [<controller>]	Teach RWIN a configuration.
LO[<port number>] [;<options>] =<text>	Load (S-records).
MD[<port number>] <address> [<count>] [<options>]	Memory display; options ;DI ;S.
M[M] <address>[;<options>]	Memory modify; options ;W ;L ;O ;V ;N ;DI.
MS <address> <data>	Memory set (also ASCII).
OF	Offset register display.
[NO]PA[<port number>]	Printer attach/detach.
PF[<port number>]	Port format.
[NO]PT	Power-Up Test enable/disable.
RESET [<SCSI-level>]	SCSI/SASI bus or SCSI controller reset.
SET	Set Date/Time
ST [<device>] [,<controller>]	Self-test
TA[<port number>]	Terminal attach
TIME	Display Date/Time
TM[<port number>] [<exit character> [<trailing character>]]	Transparent mode.
T[R] [<count>]	Trace.
TT <breakpoint address>	Trace until address.
VE[<port number>] [;<options>] =<text>	Verify (S-records).
VI	Vector Initialize (cold start).
(See description of .<register> commands)	
.A0 - .A7 [<expression>]	Display/set address register.

**TABLE 4-2. Firmware Monitor Command and Option Summary (cont'd)**

COMMAND	DESCRIPTION
.D0 - .D7 [<expression>]	Display/set data register.
.DFC        [<expression>]	Display/set destination function code.
.PC        [<expression>]	Display/set program counter.
.R0 - .R6 [<expression>]	Display/set relative offset register.
.R7	Display relative offset register.
.SFC        [<expression>]	Display/set source function code.
.SR        [<expression>]	Display/set status register.
.SS[P]      [<expression>]	Display/set supervisor stack pointer.
.US[P]      [<expression>]	Display/set user stack pointer.
.VBR        [<expression>]	Display/set vector base register.00

**Key Functions:**

(BREAK)	Abort command or process.
(DEL)	Delete character.
(CTRL-D)	Redisplay line.
(CTRL-H)	Delete character.
(CTRL-S)	Suspend output; any character continues.
(CTRL-X)	Cancel command line.
(CR) (Carriage return)	Process the current/previous command line.
* (Asterisk)	Sends any text on the current line to last port used for TM (one line transparent mode).

**CHAPTER 5****USING THE ASSEMBLER/DISASSEMBLER****5.1 INTRODUCTION**

Included as part of the MVME117 1.x debug monitor firmware is an assembler/disassembler function. The assembler/disassembler is an interactive assembler/editor in which the source program is not saved. Each source line is translated into the proper MC68010 machine language code and is stored in memory on a line-by-line basis at the time of entry. In order to display an instruction, the machine code is disassembled, and the instruction mnemonic and operands are displayed. All valid MC68010 instructions are translated.

The MVME117 debug monitor assembler is effectively a subset of the MC68010 Resident Structured Assembler. It has more limitations than the resident assembler, such as not allowing line numbers and labels; however, it is a powerful tool for creating, modifying, and debugging MC68010 code.

**5****5.1.1 M68010 Assembly Language**

The symbolic language used to code source programs for processing by the assembler is called M68010 assembly language. This language is a collection of mnemonics representing:

- . Operations
  - MC68010 machine-instruction operation codes
  - Directive (pseudo-op)
- . Operators
- . Special symbols

**5.1.1.1 Machine-Instruction Operation Codes.** That part of the assembly language that provides mnemonic machine-instruction operation codes for the MC68010 machine instructions is described in the M68000 16/32-Bit Microprocessor Programmer's Reference Manual.

**5.1.1.2 Directives.** The assembly language can contain mnemonic directives which specify auxiliary actions to be performed by the assembler. Directives are not always translated to machine language.

Assembler directives assist the programmer in:

- . Controlling the assembler output
- . Defining data and symbols
- . Allocating storage

The MVME117 debug monitor assembler recognizes only one directive called define constant (DC.W). This directive is used to define data within the program. Refer to paragraph 5.2.1.1 for a description of this directive.

### 5.1.2 Comparison with MC68000 Resident Structured Assembler

There are several major differences between the MVME117 debug monitor assembler and the MC68000 Resident Structured Assembler. The resident assembler is a two-pass assembler that processes an entire program as a unit, while the MVME117 debug monitor assembler processes each line of a program as an individual unit. Due mainly to this basic functional difference, the capabilities of the MVME117 debug monitor assembler are more restricted:

- a. Label and line numbers are not used. Labels are used to reference other lines and locations in a program. The one-line assembler has no knowledge of other program lines and, therefore, cannot make the required association between a label and the label definition located on a separate line.
- b. Source lines are not saved. In order to read back a program after it has been entered, the machine code is disassembled and then displayed as mnemonic and operands.
- c. Limited error indication. The one-line assembler will show a question mark (?) under the portion of the source statement where an error probably occurred, or will display the word "ERROR" or another short message. In contrast, the resident assembler generates specific error messages for over 60 different types of errors.
- d. Only one directive (DC.W) is accepted.
- e. No macro operation capability is included.
- f. No conditional assembly is used.
- g. Several symbols recognized by the resident assembler are not included in the MVME117 debug monitor assembler character set. These symbols include !, >, and <. Two other symbols, \* and /, each have multiple meanings to the resident assembler, depending on the context, but only one meaning to the MVME117 debug monitor assembler. Finally, the ampersand character (&) specifies a decimal number when used with the MVME117 debug monitor assembler (although numbers with no prefix are assumed to be decimal), while this symbol represents a logical AND function to the resident assembler. Paragraph 5.2.1.5 describes the MVME117 debug monitor assembler character set.

Although functional differences exist between the two assemblers, the one-line assembler is a true subset of the resident assembler. The format and syntax used with the MVME117 debug monitor assembler are acceptable to the resident assembler except as described in g. above.

## 5.2 SOURCE PROGRAM CODING

A source program is a sequence of source statements arranged in a logical way to perform a predetermined task. Each source statement occupies a line and must be either an executable instruction or a DC.W assembler directive. Each source statement follows a consistent source line format.

### 5.2.1 Source Line Format

Each source statement is a combination of operation and, as required, operand fields; line numbers, labels, and comments are not used. The general format is:

`<sp> <operation field> [<operand field>]`

The space (<sp>) must be the first character of each line. This is to be consistent with the resident assembler, which expects the first field of each line to be either a space or a label. Because the MVME117 1.x assembler never allows a label, the first character must always be a space.

**5.2.1.1 Operation Field.** The operation field must follow at least one space (more can be used) and entries can consist of one of two categories:

- a. Operation codes which correspond to the MC68010 instruction set.
- b. Define Constant directive (DC.W) which is recognized to define a constant in a word location. This is the only directive recognized by the assembler.

The size of the data field affected by an instruction is determined by the data size code. Some instructions and directives can operate on more than one data size. For these operations, the data size code must be specified or a default size applicable to that instruction will be assumed. The size code need not be specified if only one data size is permitted by the operation. The data size code is specified by a period (.), appended to the operation field, and followed by B, W, or L, where:

B = Byte (8-bit data).  
W = Word (the usual default size; 16-bit data).  
L = Longword (32-bit data).

The data size code is not permitted, however, when the instruction or directive does not have a data size attribute.

Examples (legal):

LEA	2(A0),A1	Longword size is assumed (.B, .W not allowed); this instruction loads effective address of first operand into A1.
ADD.B	(A0),D0	This instruction adds the byte whose address is (A0) to lowest order byte in D0.

ADD	D1,D2	This instruction adds low order word of D1 to low order word of D2. (W is the default size code.)
ADD.L	A3,D3	This instruction adds entire 32-bit (longword) contents of A3 to D3.

Example (illegal):

SUBA.B	#5,A1	Illegal size specification (.B not allowed on SUBA). This instruction would have subtracted the value 5 from the low order byte of A1; byte operations on address registers are not allowed.
--------	-------	--

**5.2.1.2 Operand Field.** If present, the operand field follows the operation field and is separated from the operation field by at least one space. When two or more operand subfields appear within a statement, they must be separated by a comma. In an instruction like 'ADD D1,D2' the first subfield (D1) is generally applied to the second subfield (D2) and the results placed in the second subfield. Thus, the contents of D1 are added to the contents of D2 and the result is saved in register D2. In the instruction 'MOVE D1,D2' the first subfield (D1) is the sending field and the second subfield (D2) is the receiving field. In other words, for most two-operand instructions, the general format '<opcode> <source>,<destination>' applies.

**5.2.1.3 Disassembled Source Line.** The disassembled source line may not look identical to the source line entered. The disassembler makes a decision on how to represent a numerical value based on how it interprets the number's use. If the number is determined to be an address or a "would-be" address, it is displayed in hexadecimal; everything else is decimal. For example,

MOVE.L #\$1234, \$5678

disassembles to

005000 21FC000012345678 MOVE.L #4660,\$00005678

Also, for some instructions, there are two valid mnemonics for the same opcode, or there is more than one assembly language equivalent. The disassembler may choose a form different from the one originally entered. As examples:

- a. BRA is returned for BT
- b. DBF is returned for DBRA

**NOTE**

The assembler recognizes two forms of mnemonics for two branch instructions. The BT form (branch conditionally true) has the same opcode as the BRA instruction. Also, DBRA (decrement and branch always) and DBF (never true, decrement, and branch) mnemonics are different forms for the same instruction. In each case, the assembler will accept both forms.

**5.2.1.4 Mnemonics and Delimiters.** The assembler recognizes all MC68000 instruction mnemonics except ILLEGAL. Numbers are recognized as both decimal and hexadecimal, with decimal the default case (note that this is reverse to the MVME117 1.x commands):

- a. Decimal is a string of decimal digits (0-9) without a prefix (default) or preceded by an optional ampersand (&). Examples are:

1234  
&1234

- b. Hexadecimal is a string of hexadecimal digits (0-9, A-F) preceded by a dollar sign (\$). An example is:

\$AFE5

One or more ASCII characters enclosed by apostrophes (') constitute an ASCII string. ASCII strings are left-justified and zero-filled (if necessary), whether stored or used as immediate operands. This left justification will be to a word boundary if one or two characters are specified, or to a longword boundary if the string contains more than two characters.

005000	5300	DC.W	'S'
005002	223C41424344	MOVE.L	#'ABCD',D1
005008	3536	DC.W	'56'

**NOTE**

The MC68010 has seventeen 32-bit registers (D0-D7, A0-A6, SSP, USP) in addition to a 32-bit program counter (24 bits available) and a 16-bit status register. Registers D0-D7 are used as data registers for byte, word, and longword operations. Registers A0-A6 and SSP and USP are used as software stack pointers and base address registers; they may also be used for word and longword data operations. All 17 registers may be used as index registers. Register A7 is a pseudo register, used as the system stack pointer corresponding to either SSP or USP, depending on the operating state. The MC68010 also has a vector base register and two alternate function code registers.

The following register mnemonics are recognized by the assembler:

D0-D7	Data registers.
A0-A7	Address registers.
SSP	Address register 7 represents the supervisor stack pointer of the active system state.
USP	User stack pointer. Used only in privileged instructions which are restricted to supervisory state.
CCR	Condition code register (low 8 bits of SR).
SR	Status register. All 16 bits may be modified in the supervisor state. Only low 8 bits (CCR) may be modified in user state.
PC	Program counter. Used only in forcing program counter-relative addressing.
VBR	Vector base register. Contains the 32-bit absolute address of the beginning of the exception vector.
SFC	Source function code.
DFC	Destination function code.

**5**

**5.2.1.5 Character Set.** The character set recognized by the MVME117 1.x assembler is a subset of ASCII, and these are listed below:

- a. The uppercase letters A through Z
- b. The integers 0 through 9
- c. Arithmetic operators: + -
- d. Parentheses ( )
- e. Characters used as special prefixes:

- # (pound sign) specifies the immediate form of addressing
- \$ (dollar sign) specifies a hexadecimal number
- & (ampersand) specifies a decimal number
- @ (commercial at sign) specifies an octal number
- % (percent sign) specifies a binary number
- ' (apostrophe) specifies an ASCII literal character

f. Five separating characters:

- Space
- , (comma)
- . (period)
- / (slash)
- (dash)

g. The character \* (asterisk) indicates current location.

### 5.2.2 Instruction Summary

Refer to the M68000 16/32-Bit Microprocessor Programmer's Reference Manual for descriptions of the MC68010 instructions and addressing modes.

## 5.3 ENTERING AND MODIFYING SOURCE PROGRAMS

User programs are entered into the MVME117 RAM using the one-line assembler/disassembler. The program is entered in assembly language statements on a line-by-line basis. The source code is not saved as it is converted immediately to machine code upon entry. This imposes several restrictions on the type of source line that can be entered.

Symbols and labels, other than the defined instruction mnemonics, are not allowed. The assembler has no means to store the associated values of the symbols and labels in lookup tables. This forces the programmer to use memory addresses and to enter data directly rather than use labels.

Also, editing is accomplished by retyping the entire new source line. Lines can be added or deleted by moving a block of memory data to free up or delete the appropriate number of locations.

In order to describe more clearly the procedures used to enter, modify, and execute a program, a specific example will be described. Figure 5-1 lists a program that converts an ASCII coded number into its hexadecimal equivalent. An ASCII character is in the lowest 8 bits of register D0 when the program is entered. Upon exiting, D0 contains the equivalent hexadecimal digit (0 to F), or an FF if the ASCII character does not correspond to a proper hex number.

GETHEX	CMP.B	#\$30,D0	Is hex no. < 0?
	BLT.S	ERROR	Not a hex no.
	CMP.B	#\$39,D0	Is hex no. > 9?
	BGT.S	GTHX2	
GTHX1	AND.L	#\$F,D0	Save only lower 4 bits
EXIT	BRA	*	End of routine
GTHX2	CMP.B	#\$41,D0	Is hex no. < 'A'?
	BLT.S	ERROR	Not a hex no.
	CMP.B	#\$46,D0	Is hex no. > 'F'?
	BGT.S	ERROR	Not a hex no.
	SUB.B	#7,D0	Make it smaller -- A=10
	BRA	GTHX1	
ERROR	MOVE.L	#\$FF,D0	Error code
	JMP	EXIT	

**NOTE:** Converts ASCII digit in lowest 8-bit of register D0 into hex value. Returns equivalent 0-F or FF on error in D0.

**FIGURE 5-1. Sample Program to Convert ASCII Digit to Hexadecimal Value**

For clarity, Figure 5-1 contains comments and labels. The program as it appears after entry into the MVME117 is shown in Figure 5-3. Figure 5-2 shows the ASCII character set for better understanding of the program.

b7 b6 b5 b4 b3 b2 b1 Bits					Column		0	1	2	3	4	5	6	7
					Row		0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	NUL	DLE	SP	0	@	P	\	p
0	0	0	1	1	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	0	2	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	1	3	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	0	4	4	EOT	DC4	\$	4	D	T	d	t
0	1	0	1	0	5	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	0	6	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	1	7	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	0	8	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	1	9	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	0	10	A	LF	SUB	*	:	J	Z	j	z
1	0	1	1	1	11	B	VT	ESC	+	;	K	[	k	{
1	1	0	0	0	12	C	FF	FS	,	<	L	\	l	
1	1	0	1	1	13	D	CR	GS	-	=	M	]	m	}
1	1	1	0	0	14	E	SO	RS	.	>	N	^	n	~
1	1	1	1	1	15	F	SI	US	/	?	O	_	o	DEL

**FIGURE 5-2. ASCII Character Set**

### 5.3.1 Invoking the Assembler/Disassembler

The assembler/disassembler is invoked using the ;DI option of the Memory Modify (MM) and Memory Display (MD) commands:

MM <address> ;DI

where     (CR)           sequences to next instruction  
          (.CR)          exits command

and

MD[<port number>] <address> [<count>];DI

The MM (;DI option) is used for program entry and modification. When this command is used, the memory contents at the specified location are disassembled and displayed, followed by a "?". A new or modified line can be entered if desired.

The disassembled line can be an MC68010 instruction or a DC.W directive. If the disassembler recognizes a valid form of some instruction, the instruction will be returned; if not, the DC.W \$xxxx (always hex) is returned. Because the disassembler gives precedence to instructions, a word of data that corresponds to a valid instruction will be returned as the instruction.

For the given example, the program will be entered starting at location \$2000:

```
MVME117 1.x > MM 2000;DI
001000    1005                    MOVE.B   D5,D0 ?
```

### 5.3.2 Entering a Source Line

A new source line is entered immediately following the "?", using the format discussed in paragraph 5.2.1:

```
MVME117 1.x > MM 2000;DI
002000    1005                    MOVE.B   D5,D0 ?   CMP.B   #$30,D0
```

When the carriage return is entered terminating the line, the old source line is erased from the terminal screen, the new line is assembled and displayed, and the next instruction in memory is disassembled and displayed:

```
MVME117 1.x > MM 2000;DI
002000    0C000030                CMP.B    #$30,D0
002004    FFFF                    DC.W     $FFFF ?
```

#### NOTE

If a terminal with a printer only (no CRT) is used, such as a TI 700 series device, the printer will overwrite the previous line. Therefore, a clear printout of the new entry will not be made. This also happens if the printer on port 3 is attached via the PA command.

Another program line can now be entered. Program entry continues in like manner until all lines have been entered. A period is used to exit the MM command.

If an error is encountered during assembly of the new line, the assembler will display the line unassembled with an "X" under the field suspected of causing a problem, or an error message will be displayed. Errors are discussed in paragraph 5.3.5.

### 5.3.3 Program Entry/Branch and Jump Addresses

Figure 5-3 shows the sample program as it is input to the MVME117 debug monitor one-line assembler. Notice that the comments and labels used in Figure 5-1 are not allowed; absolute addresses must be used for BRA and JMP instructions.

```

CMP.B    #$30,D0
BLT      *
CMP.B    #$39,D0
BGT      *
AND.L    #$F,D0
BRA      *
CMP.B    #$41,D0
BLT      *
CMP.B    #$46,D0
BGT      *
SUB.B    #7,D0
BRA      $200C
MOVE.L   #$FF,D0
JMP      $2012
    
```

a) First entry

```

CMP.B    #$30,D0
BLT.S    $2026
CMP.B    #$39,D0
BGT.S    $2014
AND.L    #$F,D0
BRA      *
CMP.B    #$41,D0
BLT.S    $2026
CMP.B    #$46,D0
BGT.S    $2026
SUB.B    #7,D0
BRA      $200C
MOVE.L   #$FF,D0
JMP      $2012
    
```

b) With correct branch addresses

FIGURE 5-3. Sample Program as Entered into MVME117

**5.3.3.1 Entering Absolute Addresses.** The absolute addresses are probably not known as the program is being entered. For example, when the second line is entered (BLT.S ERROR in Figure 5-1), the user does not know that the branch address (ERROR MOVE.L #\$FF,D0) will be \$2026. However, the user can instead enter an "\*" for branch to self. After the correct address (\$2026) is discovered, the second line can be reentered using the correct value. This technique can be used for forward branches and jumps. It is not required for backward branches and jumps, such as the last line of the example, because the required address is already known. If the absolute address is not within the range of a short address, a long address must be specified by appending .L to the mnemonic (BGT.L \*).

**5.3.3.2 Desired Instruction Form.** Care must be taken when entering source lines to ensure that the desired instruction form is entered. If the quick form of the instruction is wanted, it must be specified. For example:

005780 203C00000003 MOVE.L #3,D0 Assembles to the 6-byte instruction.

whereas

005780 7003 MOVEQ.L #3,D0 Assembles to the 2-byte instruction.

If the PC-relative addressing mode is desired, it must be specified. For example:

002000 41F803F0 LEA \$3F0,A0 Assembles \$3F0 as an absolute address.

whereas

002000 41FAF3EE LEA \$3F0(PC),A0 Assembles \$3F0 as a PC-relative address.

**5.3.3.3 Current Location.** To reference a current location in an operand expression, the character "\*" (asterisk) can be used. Examples are:

007000 6022 BRA \*+\$24

007000 6000FFFE BRA.L \*

007000 60FE BRA \*

#### 5.3.4 Assembler Output/Program Listings

A listing of the program is obtained using the MD command with the ;DI option. The MD command requires both the starting address and the instruction count to be entered in the command line.

Two techniques can be used to obtain a hard copy of the program using the MD command.

- a. The Printer Attach (PA) command is first used to activate the port 3, 6, 9, or A printer. An MD to the terminal will then cause a listing on the terminal and on the printer.
- b. An MD3, MD6, MD9, or MDA (MD to port 3, 6, 9, or A) command using the ;DI option will cause a listing on the printer only.

Figure 5-4 shows a listing of the sample program. Note in this example that \$E lines are specified in the MD command.

Note also that the listing does not correspond exactly to that of Figure 5-3. As discussed in paragraph 5.2.1.3, the disassembler displays in hexadecimal any number it interprets as an address; all other numbers are displayed in decimal.

```

MVME117 1.x > MD 2000 E;DI
002000      0C000030      CMP.B      #48,D0
002004      6D20         BLT.S      $002026
002006      0C000039      CMP.B      #57,D0
00200A      6E08         BGT.S      $002014
00200C      0280000000FF  AND.L      #15,D0
002012      60FE         BRA.S      $002012
002014      0C000041      CMP.B      #65,D0
002018      6D0C         BLT.S      $002026
00201A      0C000046      CMP.B      #70,D0
00201E      6E06         BGT.S      $002026
002020      04000007      SUB.B      #7,D0
002024      60E6         BRA.S      $00200C
002022      203C000000FF  MOVE.L    #255,D0
002028      4EF82012      JMP        $2012

MVME117 1.x >

```

FIGURE 5-4. Sample Program Listing

### 5.3.5 Error Conditions and Messages

There are five different conditions that can result in error messages while using the assembler/disassembler. The response to the error condition can be to abort the command (and thus the assembler), or to cause the assembler to ask for a corrected input line. The error conditions are discussed in the following paragraphs and include bus and address error traps, improper characters, numbers which are too large, and assembly errors.

**5.3.5.1 Error Traps.** Two types of errors are trapped. One form, which produces a bus error trap, may be encountered if a location is accessed where there is no memory. Included in this error type are write cycles to ROM. The second form produces an address error trap. Instructions must always begin on an even address; if not, an address error trap will result. Figure 5-5 shows examples of these conditions.

MVME117 1.x > M A00000;DI

```
2700 0000768A 8008 1105 00A00000 0000 7682 0000 FFFF 0000 6100 0557
1158 0000 00A0 0008 0000 00A0 0002 FFE1 0006 038F 4CD4 0100 0000 4CD4 0000
```

Bus Error Trap

```
PC=0000768A SR=2700=.S7.....USP=FFFFFFFF SSP=0000149A VBR=00000000 SFC=2 DFC=7
D0-7 00A00044 01964D20 FFF24D20 00000008 0000B432 00000000 00000000 00000000
A0-7 000016EA 00004EDA 000014D1 0000115C 00A00000 00001158 00001158 0000149A
PC=00768A E51C ROL.B #2,D4
```

MVME117 1.x > MM F00000;DI

```
2700 000076E8 8008 0205 00F00000 0000 8C67 0000 1419 0000 1419 051A
8C67 1280 0000 14DC 14D0 00F0 0001 FFE1 0000 0380 1419 0000 0001 1280 0001
```

Bus Error Trap

```
PC=000076E8 SR=2700=.S7.....USP=FFFFFFFF SSP=0000149A VBR=00000000 SFC=2 DFC=7
D0-7 671E8C67 00000001 00000253 00000000 0000001E 0000001E 00000002 00000000
A0-7 00001517 00F00000 000014D1 00001537 00F00000 0000153A 0000153A 0000149A
PC=0076E8 B400 CMP.B D0,D2
```

MVME117 1.x > M 10001;DI

```
2700 0000768A 800C 1105 00010001 0000 7682 0000 FFFF 0000 6100 0557
1158 0000 0001 0008 0001 0001 0003 FFE1 0006 038F 4CD4 0100 0000 4CD4 0000
```

Bus Error Trap

```
PC=0000768A SR=2700=.S7.....USP=FFFFFFFF SSP=0000149A VBR=00000000 SFC=2 DFC=7
D0-7 00010044 01964D20 FFF24D20 00000008 0000B432 00000000 00000000 00000000
A0-7 000016EA 00004EDA 000014D1 0000115C 00010001 00001158 00001158 0000149A
PC=00768A E51C ROL.B #2,D4
```

FIGURE 5-5. Examples of Error Traps

Also note that bus and address errors also cause display of the exception status from the stack, in hexadecimal characters.

For details on this display, refer to the bus error and address error descriptions in the M68000 16/32-Bit Microprocessor Programmer's Reference Manual.

**5.3.5.2 Improper Character.** If a character appears in the operand field that does not belong to the class of characters specified or expected, an "X" will be printed beneath the character string suspected of containing the improper character, followed by a "?" to prompt reentry of the line. For example, if a % (percent sign) is used to specify the binary class of characters, only the digits 0 and 1 will be accepted.

```

MVME117 1.x > MM 6000;DI      S is not a decimal digit
006000      FFFF      DC.W      $FFFF ?  MOVE.W  #S',DO
006000      MOVE.W      #S',DO
                                X?

MVME117 1.x > MM 6000;DI      9 is not an octal digit
006000      FFFF      DC.W      $FFFF ?  ADDA.L  #@974,A6
006000      ADDA.L      #@974,A6
                                X?

MVME117 1.x > MM 6000;DI      P is not a decimal digit
006000      FFFF      DC.W      $FFFF ?  JMP  $4000+PC
006000      JMP      $4000+PC
                                X?
    
```

FIGURE 5-6. Examples of Improper Characters

**5**

**5.3.5.3 Number Too Large.** Another error type involves numbers which are too large for the MC68010 to handle. Again, an "X" is printed under the number suspected of containing the error, followed by a "?". Figure 5-7 gives an example.

```

MVME117 1.x > MM 4000;DI      Value is larger than 32 bits
004000      FFFF      DC.W      $FFFF ?  LEA.L  $937402110,A7
004000      LEA.L      $937402110,A7
                                X?
    
```

FIGURE 5-7. Example of a Number Which Is Too Large

**5.3.5.4 Assembly Errors.** An assembly error can occur due to an invalid opcode, an illegal addressing mode for a particular instruction, a format which is in error (leading space omitted as an example), or a source line which is incorrect in some other way. When the entry as written is not a valid MC68010 instruction, the assembler echoes the source line up to and including the field in which the error probably occurred. It also prints an "X" under the field suspected of containing an error, followed by a "?" to prompt reentry of the line.

The entire line must be reentered in its correct form. If the error has not been corrected or another is encountered, the error indicator will be returned. After all errors have been corrected and the source line represents a valid MC68010 instruction, the line will be assembled. The memory address, machine code, and source code will be displayed and the next line will be disassembled. A period (.) is used to exit the command. Examples of typical errors are shown in Figure 5-8.

**Example 1 Invalid Opcode**

006700	FFFF	DC.W	\$FFFF ?	<u>BEQU.S</u>	<u>\$6754</u>
		BEQU.S			
		X?	<u>BEQ.S</u>	<u>\$6754</u>	
006700	6752	BEQ.S	\$6754		

**Example 2 Missing Leading Space**

001100	FFFF	DC.W	\$FFFF ?	<u>OR.B</u>	<u>D5,(A6)</u>
		X?	<u>OR.B</u>	<u>D5,(A6)</u>	
001100	8B16	OR.B	D5,(A6)		

**Example 3 Unrecognizable Opcode**

005300	FFFF	DC.W	\$FFFF ?	<u>MULSW</u>	<u>52,D3</u>
		MULSW			
		X?	<u>MULS.W</u>	<u>52,D3</u>	
005300	C7F80034	MULS.W	52,D3		

**Example 4 Invalid Size Extension**

007200	FFFF	DC.W	\$FFFF ?	<u>MOVEQ.B</u>	<u>#2,D1</u>
		MOVEQ.B	#2,D1		
		X?	<u>MOVEQ.L</u>	<u>#2,D1</u>	
007200	7202	MOVEQ.L	#2,D1		

**Example 5 Invalid Addressing Mode**

001500	FFFF	DC.W	\$FFFF ?	<u>ADDQ.B</u>	<u>#7,A0</u>
		ADDQ.B	#7,A0		
		X?	<u>ADDQ.B</u>	<u>#7,(A0)</u>	
001500	5E10	ADDQ.B	#7,(A0)		

**Examples 6 and 7 Branch Address Too Large**

004900	FFFF	DC.W	\$FFFF ?	<u>BRA</u>	<u>\$10000</u>
		BRA	\$10000		
		X?	<u>BRA</u>	<u>\$8000</u>	
004900	600036FE	BRA	\$8000		
004800	FFFF	DC.W	\$FFFF ?	<u>BRA.S</u>	<u>\$7000</u>
		BRA.S	\$7000		
		X?	<u>BRA.S</u>	<u>\$4902</u>	
		BRA.S	\$4902		
		X?	<u>BRA.S</u>	<u>\$4860</u>	
004800	605E	BRA.S	\$4860		

FIGURE 5-8. Examples of Assembly Errors

**5**

THIS PAGE INTENTIONALLY LEFT BLANK.

**CHAPTER 6****MVME117 FIRMWARE MONITOR ROUTINES****6.1 INTRODUCTION**

There are two general categories of support provided by the TRAP #15 routines. The first is the general purpose I/O and conversion type routine that has been available in Motorola debug monitors previously: "input/output a line", "convert to/from hex", "return to the firmware monitor", etc. These functions provide convenient debugging tools in a test environment, but are not intended to be replacements for operating systems with their more powerful program development tools. The user should be aware that these are the very routines used within the firmware monitor, and must be informed of the linkage required to use them.

The second category of support as provided with the MVME117 debug firmware is high level SCSI bus access. Contained within the debug monitor EPROMs are the detailed low level SCSI bus routines that arbitrate for the SCSI bus, provide the detailed handshaking, and actually transfer the data to or from specific controllers. This low level support is provided by SCSI firmware, and is accessible through TRAP #15 calls. Although SCSI firmware resides within the same two EPROM's as the firmware debug monitor, it is not dependent upon the monitor in any way. (SCSI firmware was designed to support user-written monitors and operating systems not using Motorola's debug monitor.)

**NOTE**

Although specific support for the MC68881 is not provided either in TRAP #15 or by the debug monitor in general, the user may find information on its use as a peripheral in the document MC68881PER (MC68881 Floating Point Coprocessor as a Peripheral in an MC68000/008/010/012 System.)

**6.2 USER I/O THROUGH TRAP #15**

When the TRAP #15 handler gains control, a short stack frame has been placed on the stack (as a result of handling the TRAP exception). The two-byte number specified in the DC.W is extracted and examined to select the function requested. For SCSI requests, the packet address has been examined and removed from the stack. Once the function has been extracted and validated, control is passed to the appropriate routine. Upon completion the registers are restored and control returned to the user through the use of an RTE instruction. Control is passed back to the user at the instruction following the DC.W. For SCSI functions, (\$10 - \$19), status byte 1 from the packet is placed into register DO.B with the status register reflecting the value of DO, allowing the user to use a BEQ for good status (or a BNE for bad), following a TRAP #15 SCSI call.

**NOTE**

For a detailed list of specific SCSI error codes, refer to Appendix E.

### 6.2.1 TRAP #15 Summary

#### TRAP #15 SCSI

<u>Function</u>	<u>Code</u>	<u>Description of Function</u>	<u>Parameters Required</u>
\$00	N/A	Return to firmware monitor	None
\$01	N/A	Read from console	A5 & A6 = Addr. of buffer
\$02	N/A	Write to console	A5=Start, A6=End+1
\$03	N/A	Read from any port	D0=Port#, A5 & A6 = buffer
\$04	N/A	Write to any port	D0=Port#, A5=Start, A6=End+1
\$05	N/A	Write to console - NO CR/LF	A5=Start, A6=End+1
\$06	N/A	Write to any printer - NO CR/LF	D0=Port#, A5=Start, A6=End+1
\$07	N/A	Convert ASCII to hex	A5=Addr. of string, D0=Result
\$08	N/A	Convert 1 hex digit to ASCII	D0=Source, result in (A6)+
\$09	N/A	Convert 2 hex digits to ASCII	D0=Source, result in (A6)+
\$0A	N/A	Convert 4 hex digits to ASCII	D0=Source, result in (A6)+
\$0B	N/A	Convert 6 hex digits to ASCII	D0=Source, result in (A6)+
\$0C	N/A	Convert 8 hex digits to ASCII	D0=Source, result in (A6)+
\$0D		Reserved	
\$0E		Reserved	
\$0F		Reserved	
\$10	\$00	Read SCSI drive/controller	Push addr. of packet on stack
\$11	\$04	Write SCSI drive/controller	Push addr. of packet on stack
\$12	\$08	Attach SCSI drive/controller	Push addr. of packet on stack
\$13	\$0C	Detach SCSI drive/controller	Push addr. of packet on stack
\$14	\$10	Format (with/without defect list)	Push addr. of packet on stack
\$15	\$14	Assign alternate sector (SCSI)	Push addr. of packet on stack
\$16	\$18	Assign alternate track (SASI)	Push addr. of packet on stack
\$17	\$1C	Custom SCSI Sequence	Push addr. of packet on stack
\$18	\$20	SCSI bus reset	Push addr. of packet on stack
\$19	\$24	SCSI controller reset	Push addr. of packet on stack

#### NOTE

The D0 parameter for functions 3, 4, and 6 indicates that the port number to be selected must be placed into D0 prior to issuing the TRAP #15 instruction. (Refer to paragraph 6.2.2.2 for a description of the serial and printer ports.) Previous TRAP #15 handlers provided functions for every port available for input or output. Now there are only six calls which support console and general terminal I/O, each of the output calls being able to select optional carriage return and line feed characters to be inserted at the end of the line.

### 6.2.2 TRAP #15 User Interface

**6.2.2.1 Normal TRAP #15 Calls.** Sample TRAP #15 calls with their parameters follow:

**\$00 Return to Firmware Monitor**

TRAP	#15	Initiate the TRAP #15 interrupt handler
DC.W	\$0	Return to Firmware Monitor

**\$01 Input from the console**

LEA	AREA,A5	A5 = Address of input area
MOVE.L	A5,A6	A6 = Address of input area
TRAP	#15	Initiate the TRAP #15 interrupt handler
DC.W	\$1	From console (with echo)

Upon return: A5 will point to the start of the input AREA  
A6 will point to the last byte entered + 1

**\$02 Output to console (with CR/LF)**

LEA	MSG1,A5	A5 = Start of message (must be RAM to insert CR/LF)
LEA	MSG1E,A6	A6 = End of message + 1
TRAP	#15	Initiate the TRAP #15 interrupt handler
DC.W	\$2	To console (with CR/LF)

**\$03 Input from any port (no echo)**

MOVE.L	#5,D0	D0 = Port #5 (MVME050 serial port2)
LEA	AREA,A5	A5 = Address of input area
MOVE.L	A5,A6	A6 = Address of input area
TRAP	#15	Initiate the TRAP #15 interrupt handler
DC.W	\$3	From any port (no echo)

Upon return: A5 will point to the start of the input AREA  
A6 will point to the last byte entered + 1

**\$04 Output to any port (with CR/LF)**

MOVE.L	#6,D0	D0 = Port #6 (MVME050 printer port)
LEA	MSG1,A5	A5 = Start of message (must be RAM to insert CR/LF)
LEA	MSG1E,A6	A6 = End of message + 1
TRAP	#15	Initiate the TRAP #15 interrupt handler
DC.W	\$4	To any port (with CR/LF)

**\$05 Output to console (no CR/LF)**

LEA	MSG1,A5	A5 = Start of message
LEA	MSG1E,A6	A6 = End of message + 1
TRAP	#15	Initiate the TRAP #15 interrupt handler
DC.W	\$5	To console (no CR/LF)

\$06     Output to any printer (no CR/LF)

```

MOVE.L #9,D0      D0 = Port #9 (MVME410 printer port A)
LEA     MSG1,A5    A5 = Start of message
LEA     MSG1E,A6   A6 = End of message + 1
TRAP    #15        Initiate the TRAP #15 interrupt handler
DC.W    $6         Any printer (no CR/LF)

```

\$07     Convert ASCII string to 32-bit hex number

```

LEA     MSG1,A5    A5 = Start of ASCII string
LEA     MSG1E,A6   A6 = End of ASCII string + 1
TRAP    #15        Initiate the TRAP #15 interrupt handler
DC.W    $7         Call GETNUMA

```

Upon return: D0 will contain the 32-bit hex number

\$08     Convert 1-digit hex number in D0, place into ASCII string at (A6)+

```

LEA     MSG1,A5    A5 = Start of ASCII string
LEA     MSG1E,A6   A6 = End of ASCII string + 1
TRAP    #15        Initiate the TRAP #15 interrupt handler
DC.W    $8         Call PUTHEX

```

Upon return: The string pointed to by A5 and A6 will have the ASCII byte appended to the back of the original string, with A6 post incremented.

\$09     Convert 2-digit hex number in D0, place into ASCII string at (A6)+

```

LEA     MSG1,A5    A5 = Start of ASCII string
LEA     MSG1E,A6   A6 = End of ASCII string + 1
TRAP    #15        Initiate the TRAP #15 interrupt handler
DC.W    $9         Call PNT2HX

```

Upon return: The string pointed to by A5 and A6 will have the two ASCII bytes appended to the back of the original string, with A6 post incremented.

\$0A     Convert 4-digit hex number in D0, place into ASCII string at (A6)+

```

LEA     MSG1,A5    A5 = Start of ASCII string
LEA     MSG1E,A6   A6 = End of ASCII string + 1
TRAP    #15        Initiate the TRAP #15 interrupt handler
DC.W    $A         Call PNT4HX

```

Upon return: The string pointed to by A5 and A6 will have the four ASCII bytes appended to the back of the original string, with A6 post incremented.

**\$0B** Convert 6-digit hex number in D0, place into ASCII string at (A6)+

```

LEA    MSG1,A5      A5 = Start of ASCII string
LEA    MSG1E,A6     A6 = End of ASCII string + 1
TRAP   #15          Initiate the TRAP #15 interrupt handler
DC.W   $B           Call PNT6HX

```

Upon return: The string pointed to by A5 and A6 will have the six ASCII bytes appended to the back of the original string, with A6 post incremented.

**\$0C** Convert 8-digit hex number in D0, place into ASCII string at (A6)+

```

LEA    MSG1,A5      A5 = Start of ASCII string
LEA    MSG1E,A6     A6 = End of ASCII string + 1
TRAP   #15          Initiate the TRAP #15 interrupt handler
DC.W   $C           Call PNT8HX

```

Upon return: The string pointed to by A5 and A6 will have the eight ASCII bytes appended to the back of the original string, with A6 post incremented.

#### 6.2.2.2 MVME117 Firmware Port Assignments

PORT  
NUMBER

DESCRIPTION

none	Default	MVME117 port 1 (connector on MVME117)
1	Specifies	MVME117 port 1 (connector on MVME117)
2	Specifies	MVME117 port 2 (connector on MVME708-1 Transition Module)
3	Specifies	MVME117 printer (connector on MVME708-1 Transition Module)
4	Specifies	MVME050 terminal 1 (connector on MVME701 Transition Module)
5	Specifies	MVME050 terminal 2 (connector on MVME701 Transition Module)
6	Specifies	MVME050 printer (connector on MVME701 Transition Module)
7	Specifies	MVME400 terminal 2 (connector on MVME400)
8	Specifies	MVME400 terminal 1 (connector on MVME400)
9	Specifies	MVME410 printer A (connector on MVME410)
A	Specifies	MVME410 printer B (connector on MVME410)

**6.2.2.3 SCSI TRAP #15 Calls.** All TRAP #15 SCSI functions require that an appropriate packet be constructed and that the address of the packet be placed on the stack prior to invoking TRAP #15. Upon return from TRAP #15, D0.B contains status byte 1 in the packet whose contents are reflected in the status register of the MPU. (Refer to Appendix E for packet formats.)

**\$10** Read sector(s) from SCSI

```

PEA    READWRIT     Push address of packet on stack
TRAP   #15          Invoke TRAP #15
DC.W   $10          Call SCSI Read
BNE    ERROR        Go to Error if not "OK"

```

**\$11 Write sector(s) to SCSI**

PEA	READWRIT	Push address of packet on stack
TRAP	#15	Invoke TRAP #15
DC.W	\$11	Call SCSI Write
BNE	ERROR	Go to Error if not "OK"

**\$12 Attach a controller/drive**

PEA	ATTACHP	Push address of packet on stack
TRAP	#15	Invoke TRAP #15
DC.W	\$12	Call SCSI Attach
BNE	ERROR	Go to Error if not "OK"

**\$13 Detach a controller/drive**

PEA	ATTACHP	Push address of packet on stack
TRAP	#15	Invoke TRAP #15
DC.W	\$13	Call SCSI Detach
BNE	ERROR	Go to Error if not "OK"

**\$14 Format (with/without defect list)**

PEA	FORMATP	Push address of packet on stack
TRAP	#15	Invoke TRAP #15
DC.W	\$14	Call SCSI Format
BNE	ERROR	Go to Error if not "OK"

Refer to Appendix E for format of optional defect list.

**\$15 Assign alternate sector (SCSI)**

PEA	ASSIGNSP	Push address of packet on stack
TRAP	#15	Invoke TRAP #15
DC.W	\$15	Call SCSI Assign Alternate Sector
BNE	ERROR	Go to Error if not "OK"

Refer to Appendix E for format of defect list.

**\$16 Assign alternate track (SASI)**

PEA	ASSIGNTP	Push address of packet on stack
TRAP	#15	Invoke TRAP #15
DC.W	\$16	Call SCSI Assign Alternate Track
BNE	ERROR	Go to Error if not "OK"

**\$17 Custom SCSI sequence**

PEA	CUSTOMP	Push address of packet on stack
TRAP	#15	Invoke TRAP #15
DC.W	\$17	Call SCSI Custom Sequence
BNE	ERROR	Go to Error if not "OK"

Refer to Appendix E for format of additional control blocks required.

**\$18 SCSI bus reset**

PEA	BRESETP	Push address of packet on stack
TRAP	#15	Invoke TRAP #15
DC.W	\$18	Call SCSI Bus Reset
BNE	ERROR	Go to Error if not "OK"

**\$19 SCSI controller reset**

PEA	ASSIGNP	Push address of packet on stack
TRAP	#15	Invoke TRAP #15
DC.W	\$19	Call SCSI Controller Reset
BNE	ERROR	Go to Error if not "OK"

**6.2.2.4 Example TRAP #15 Program.** The following example program shows sample calls requesting terminal and SCSI I/O.

**EXAMPLE PROGRAM:**

**COMMENT**

```

1          00002000          ORG      $2000
2
3 00002000 4BFA0040  LOOP:  LEA      MSG1(PC),A5      A5 = Start of Message
4 00002004 4DFA004F          LEA      END1(PC),A6      A6 = End of Message+1
5 00002008 4E4F          TRAP      #15      Invoke TRAP 15
6 0000200A 0002          DC.W      2          Specifying "Output to Console (CR-LF)"
7
8 0000200C 4E4F          DEBUG: TRAP      #15      Invoke TRAP 15...
9 0000200E 0000          DC.W      0          Specifying "Return to Firmware monitor"
10
11 00002010 4B790000207A  PEA      ATTACHP      Load address of Attach packet
12 00002016 4E4F          TRAP      #15      Invoke TRAP 15...
13 00002018 0012          DC.W      $12      * Specifying "ATTACH"
14 0000201A 6618          BNE.S    BADSCSI      Trap #15 complete OK?
15
16 0000201C 4B790000209E  PEA      READP      Place Address of packet onto STACK
17 00002022 4E4F          TRAP      #15      Invoke TRAP 15
18 00002024 0010          DC.W      $10      * Specifying "READ" Sector 0 for 10
19 00002026 660C          BNE.S    BADSCSI      Trap #15 complete OK?,
20
21 00002028 4B790000207A  PEA      ATTACHP      Place Address of packet onto STACK
22 0000202E 4E4F          TRAP      #15      Invoke TRAP 15
23 00002030 0013          DC.W      $13      * Specifying "DETACH"
24 00002032 67CC          BEQ      LOOP      Good Status... do it again
25
26          *
27          * Disk error routine
28          *
29 00002034 4BFA0022  BADSCSI: LEA      BADIOM(PC),A5      A5 = Start of Message
30 00002038 4DFA003C          LEA      BADIOE(PC),A6      A6 = End of Message+1
31 0000203C 4E4F          TRAP      #15      Invoke TRAP 15
32 0000203E 0002          DC.W      2          Specifying "Output to Console (CR-LF)"
33 00002040 60CA          BRA      DEBUG      Return to debugger
34
35          *
36          * Constants and variables
37          *
38          0000000D  CR      EQU      $0D
39          0000000A  LF      EQU      $0A
40 00002042 0D0A456E7465 MSG1:  DC.B      CR,LF,'Enter 60 to begin'
41 00002055 202020          END1:  DC.B      ' '
42 0000205B 204261642053 BADIOM: DC.B      ' Bad SCSI status (See DO...)',CR,LF
43 00002076 202020          BADIOE: DC.B      ' '
44

```

```

45      *
46      * SCSI Packets used to Attach, Read and Detach
47      *
48      0000207A 00000000  ATTACHP  DS.W    $0      * A T T A C H / D E T A C H *
49      0000207A 01          DC.B    $1      Controller
50      0000207B 00          DC.B    $0      Drive
51      0000207C 0000        DC.B    $0,$0   Status
52      0000207E 0000        DC.B    $0,$0   Step Rate
53      00002080 0006        DC.B    $0,$6   Number of heads
54      00002082 0132        DC.W    $132   Number of Cylinders
55      00002084 0099        DC.W    $99    Precomp Cylinder
56      00002086 0021        DC.W    $21    Sectors per Track
57      00002088 0000        DC.W    $0     .filler
58      0000208A 0050        DC.W    $50    Controller Code (Buf, Hard Disk)
59      0000208C 0000        DC.W    $0     .filler
60      0000208E 00          DC.B    $0     .filler
61      0000208F 00          DC.B    $00    SCSI Function (Provided by TRAP 15)
62      00002090 0046        DC.B    $0,$46 Level and Vector (0 = Polled Mode)
63      00002092 0000        DC.W    $0     Status bytes 3 & 2
64      00002094 80          DC.B    $80    Controller type/Drive type
65      00002095 05          DC.B    $05    Retry count
66      00002096 0100        DC.W    $100   Sector Size
67      00002098 00          DC.B    $0     .filler
68      00002099 0C          DC.B    $C     Number Alt Cyls to reserve
69      0000209A 00002088    DC.L    WORK   Address of RAM work area
70
71      0000209E 00000000  READP   DS.W    $0      * R E A D *
72      0000209E 01          DC.B    $1      Controller
73      0000209F 00          DC.B    $0      Drive
74      000020A0 0000        DC.B    $0,$0   Status bytes 0 & 1
75      000020A2 00003000    DC.L    $3000   I/O Area Address
76      000020A6 00000000    DC.L    $0     Starting Sector Number
77      000020AA 0010        DC.W    $10    Number of Sectors
78      000020AC 0000        DC.W    $0     .filler
79      000020AE 0000        DC.W    $0     .filler
80      000020B0 0000        DC.W    $0     .filler
81      000020B2 00          DC.B    $0     .filler
82      000020B3 00          DC.B    $0     SCSI Function (Provided by Trap 15)
83      000020B4 0046        DC.B    $0,$46 Level and Vector (0 = Polled Mode)
84      000020B6 0000        DC.W    $0     Status bytes 2 & 3
85
86      000020B8 000000A0  WORK    DS.B    160   RAM Work area required by SCSI Firmware
87
88      00002158 0000018C          DS.L    99   Stack area
89      000022E4 00000004  STACK   DS.L    1
90
91      END

```

```

***** TOTAL ERRORS      0--
***** TOTAL WARNINGS    0--

```

### 6.3 MVME117 FIRMWARE MONITOR SUBROUTINES

A branch table is located at the beginning of ROM to allow use of Debug Monitor subroutines under various operating systems. Note, however, that TRAP #15 support is the standard interface between the user and I/O routines. Use of the branch table should be based upon a complete understanding of the MVME117 Firmware Monitor 1.x source release.

The branch table allows a calling routine to access any of the supported subroutines with a BSR to a long branch located at the start of EPROM. The user will always be able to access these subroutines despite future changes in their locations because the table offset will not change (even though the actual subroutine addresses might).

In most cases, the following subroutines can be entered with a BSR to the address shown for each entry.

#### \$F0000C TRACE -- Trace one instruction

No registers are required for linkage except the stack for the RTS.

#### \$F00010 BOCMD -- Disk boot entry point

No registers are required for linkage.

#### \$F00014 ABORTB -- Software abort routine

No registers are required for linkage; control is returned to the firmware monitor.

#### \$F00018 CHKBP -- Illegal instruction vector

No registers are required for linkage; control is returned to the firmware monitor.

#### \$F00020 DISKR -- Disk read routine

A0	Address of hardware (controller/drive)
A2	Address of RAM configuration area for this controller/drive
D0	Number of 256-byte blocks
D2	Memory address
D3	Block number
IPCNUM	RAM variable containing 1-byte controller number
DRVNUM	RAM variable containing 1-byte drive number

\$F00024 DISKW -- Disk write routine

A0        Address of hardware (controller/drive)  
A2        Address of RAM configuration area for this controller/drive  
D0        Number of 256-byte blocks  
D2        Memory address  
D3        Block address  
IPCNUM    RAM variable containing 1-byte controller number  
DRVNUM    RAM variable containing 1-byte drive number

\$F00028 INITVECT -- Initialize vectors #2 through #255\$F00040 COLDSTRT -- User requests reset of VMEbus and onboard parts

If the vectors and RAM semaphore are still valid, a warm start will be executed. Otherwise, a cold start will be executed. Care should be taken to disable interrupts by placing an \$FF into the MVME117 Module Control Register (MCR) at \$F44007 before branching to this address.

\$F00048 SCSI1 -- SCSI ROM entry for Command Entry

A2        Address of SCSI packet  
Entry via JMP  
Return via vector specified in packet

\$F0004C SCSI1 -- SCSI ROM entry for Reactivation

No entry parameters  
Entry via JMP  
Return via vector specified in packet

\$F00050 SCSI1 -- SCSI ROM entry for Interrupt Service

No entry parameters  
Entry through hardware vector #44  
Return via vector specified in packet at command entry time (Attach)

THIS PAGE INTENTIONALLY LEFT BLANK.

## APPENDIX A

### SOFTWARE ABORT

If a target program must be stopped with the stack data preserved, the user may press the ABORT switch (see Figure 2-2). This will generate a level 7 interrupt vector which will interrupt the target program and load the contents of RAM location \$7C into the program counter. If the default vector locations have not been overwritten, the console will display Software Abort and the following data will be saved: address registers, data registers, program counter, status register, supervisor stack pointer, user stack pointer, vector base register, destination function code, and source function code.

Remember that the firmware monitor shares resources with the target program under test (refer to paragraph 1.3.2). Therefore, if the target program changes the contents of location \$7C, this abort feature is lost.

In contrast to the abort feature, the contents of the target supervisor stack pointer, program counter, and status register are lost when the RESET switch is pressed. The RESET feature sets the processor to supervisor state, loads the supervisor stack pointer with the contents of RAM locations 0-\$3, and loads the program counter with the contents of RAM locations \$4-\$7. It also saves the contents of the target registers for display by the Display Formatted Registers (DF) command.

THIS PAGE INTENTIONALLY LEFT BLANK.

## APPENDIX B

### FIRMWARE MONITOR MESSAGES

<u>ERROR MESSAGE</u>	<u>MEANING</u>
DATA DID NOT STORE	Data did not go where intended (such as attempting to write to ROM).
DISK ERROR: Status=xx xx xx xx xx DISK ERROR: Status=Busy	xx xx xx xx xx Refer to the Winchester Disk Controller User's Manual for explanation of the ten status bytes.
Error	Returned if GT operand is the same as Breakpoint set previously.  Returned if Hexadecimal address contains non-hex digits.
<type> Error Trap	Refer to TRAPS in M68000 16/32-Bit Microprocessor Programmer's Reference Manual.
Foreign Media - Command aborted	EXORMACS or MOTOROLA not contained within the volume ID at locations \$F8-\$FF.
ILLEGAL INSTRUCTION	Instruction used an illegal opcode.
Invalid Address	Address too big (1 in bits 24 - 31) or odd for .W or .L (1 in bit 0).
Invalid Option... valid options are... ;DI or ;S	Memory Display command response to invalid option.
IS NOT A HEX DIGIT	Improper character entered in a field that requires a hexadecimal digit.
NOT HEX	Same as IS NOT A HEX DIGIT.
Offset Vector \$xxx Error Trap	Indicates uninitialized vector.
PRINTER NOT READY	Printer is not properly connected or cannot receive output.
What?	Program does not recognize user's entry.

<u>DIAGNOSTIC ERROR MESSAGES</u>	<u>MEANING</u>
68010 MPU Test PASSED	Test passed.
68881 Test PASSED	Test passed.
Abort Button Test PASSED	Test passed.
CMOS Memory Test (Size = xx) PASSED	Test passed.
Disk/Controller Test PASSED	Test passed.
Front Panel Switch Test PASSED	Test passed.
LED Test PASSED	Test passed.
Parity Logic Test PASSED	Test passed.
Parity RAM Test PASSED	Test passed.
PTM Test PASSED	Test passed.
RAM Test - Parity Disabled (Size = xxxx) PASSED	Test passed.
RAM Test - Parity Enabled PASSED	Test passed.
Real Time Clock Test PASSED	Test passed.
ROM Test PASSED	Test passed.
SCSI Test PASSED	Test passed.
SELF-TEST PASSED	The self-test detected no failures.
Serial Port Test This is a test of Port #x PASSED	Test passed.
Serial Port Test FAILED Error Code \$xxxx	Error code resulting from Serial Port Test failure. Refer to paragraph 4.2.34 for description.

## OTHER MESSAGES

## MEANING

At Breakpoint	Indicates program has stopped at breakpoint.
Autoboot Sequence Started...	Software Abort to Interrupt Message displayed at power-up informing user that Autoboot will complete unless interrupted by Software Abort.
Booting from: xxxx	Indicates the volume ID of the disk being booted. Message is suppressed if volume ID is null.
Break	BREAK key has been used.
COLD Start	Vectors have been loaded.
WARM Start	Vectors have not been loaded.
End Onboard RAM = 01FFFF Start Offboard RAM = 020000 End Offboard RAM = 09FFFF	Following warm or cold start, this message displays RAM boundaries.
MVME117 1.x >	Firmware monitor prompt.
PARTIAL dump completed	Too few sectors were allocated on the disk to allow a complete memory dump. As much memory as fits is dumped to the disk.
Physical Address=<address>	Actual address calculated using parameters and relative offsets.
.PC within "DEBUGGER"	Displayed by trace commands indicating care must be taken while "TESTING" within the firmware monitor (e.g., with breakpoints and Stack).
Size and Initialize RAM (Y/N) ?	Prompt presented at power-up (no Autoboot) or following the VI command.
Software Abort	Displayed when ABORT switch is used.

B

**B**

THIS PAGE INTENTIONALLY LEFT BLANK.

## APPENDIX C

### CONFIGURATION AREA

Disks initialized using some systems contain a Volume Identification Block and a Disk Configuration Block in sectors 0 and 1, respectively. The firmware monitor looks for either "EXORMACS" or "MOTOROLA" in locations \$F8-\$FF of sector 0 to validate the disk. Refer to paragraph 1.3.3 for more information used from the volume ID. The firmware monitor then uses the following parameters from the Disk Configuration Block to access the disk:

- Attributes word
- Physical sectors per track on media
- Number of heads on drive
- Number of cylinders on media
- Physical sector size of media
- Precompensation cylinder number

The complete Disk Configuration Block is shown below:

<u>256-BYTE SECTOR 1 OFFSET</u>	<u>LENGTH IN BYTES</u>	<u>PARAMETER DESCRIPTION</u>
0	1	Status/Error
1	1	Channel Type Code
2	1	Device Type Code
3	1	Driver Code
4	2	Attributes Mask
6	2	Parameters Mask
8	2	Attributes Word
10(\$A)	2	VERSA dos Sector Size
12(\$C)	4	Total Number of VERSA dos Sectors
16(\$10)	4	WRITE Time-out (Unused)
20(\$14)	4	READ Time-out (Unused)
24(\$18)	1	Physical Sectors per Track on Media
25(\$19)	1	Number of Heads on Drive
26(\$1A)	2	Number of Cylinders on Media
28(\$1C)	1	Interleave Factor on Media
29(\$1D)	1	Spiral Offset on Media
30(\$1E)	2	Physical Sector Size on Media
32(\$20)	2	Starting Head Number of Drive
34(\$22)	2	Number of Cylinders on Drive
36(\$24)	2	Precompensation Cylinder # (usually .5 total cyl)
38(\$26)	1	Physical Sectors per Track on Drive
39(\$27)	1	Stepping Rate Code
40(\$28)	2	Reduced WRITE Current Cylinder Number
42(\$2A)	2	ECC Data Burst Length
44(\$2C)	2	Reserved

Disks initialized on some systems may not contain the Volume Identification Block or the Disk Configuration Block. These disks cannot be accessed by the firmware monitor until the locations \$F8-\$FF of sector 0 are modified to contain either "EXORMACS" or "MOTOROLA". The firmware monitor then uses the following default values to access the disk. These default values will allow access of track 0 for all configurations.

<u>PARAMETER DESCRIPTION</u>	RWIN	
	<u>FLOPPY DISK</u>	<u>HARD DISK</u>
Attributes word	\$0F	\$10
Physical sectors per track on media	10	N/A
Number of heads on drive	2	1
Number of cylinders on media	50	1
Physical sector size of media	N/A	N/A
Precompensation cylinder number	N/A	0

The attributes word is defined as:

Bit 7 / 0 / Bit 6 / 0 / Bit 5 / 0 / Bit 4 / MT / Bit 3 / SN / Bit 2 / DS / Bit 1 / MF / Bit 0 / TD /

MT - Media Type  
0 = Floppy disk  
1 = Hard disk

SN\* - Sector Numbering  
0 = Motorola format  
1 = IBM format

DS\* - Diskette Sides  
0 = Single sided  
1 = Double sided

MF\* - Recording Method  
0 = Single data density (FM)  
1 = Double data density (MFM)

TD\* - Track Density  
0 = Single track density (48 TPI)  
1 = Double track density (96 TPI)

\* Floppy disk attribute only.

## APPENDIX D

### S-RECORD OUTPUT FORMAT

The S-record format for output modules was devised for the purpose of encoding programs or data files in a printable format for transportation between computer systems. The transportation process can thus be visually monitored and the S-records can be more easily edited.

#### S-RECORD CONTENT

When viewed by the user, S-records are essentially character strings made of several fields which identify the record type, record length, memory address, code/data, and checksum. Each byte of binary data is encoded as a 2-character hexadecimal number: the first character representing the high-order four bits, and the second the low-order four bits of the byte.

The five fields which comprise an S-record are shown below:

type	record length	address	code/data	checksum
------	---------------	---------	-----------	----------

where the fields are composed as follows:

<u>FIELD</u>	<u>PRINTABLE CHARACTERS</u>	<u>CONTENTS</u>
type	2	S-record type -- S0, S1, etc.
record length	2	The count of the character pairs in the record, excluding the type and record length.
address	4, 6, or 8	The 2-, 3-, or 4-byte address at which the data field is to be loaded into memory.
code/data	0-2n	From 0 to n bytes of executable code, memory-loadable data, or descriptive information. For compatibility with teletypewriters, some programs may limit the number of bytes to as few as 28 (56 printable characters in the S-record).
checksum	2	The least significant byte of the one's complement of the sum of the values represented by the pairs of characters making up the record length, address, and the code/data fields.

When downloading S-records, each record must be terminated with a CR. Additionally, an S-record may have an initial field to accommodate other data such as line numbers generated by some time-sharing systems.

Accuracy of transmission is ensured by the record length (byte count) and checksum fields.

### S-RECORD TYPES

Eight types of S-records have been defined to accommodate the several needs of the encoding, transportation, and decoding functions. The various Motorola upload, download, and other record transportation control programs, as well as cross assemblers, linkers, and other file-creating or debugging programs, utilize only those S-records which serve the purpose of the program. For specific information on which S-records are supported by a particular program, the user's manual for that program must be consulted.

An S-record-format module may contain S-records of the following types:

- S0 The header record for each block of S-records. The code/data field may contain any descriptive information identifying the following block of S-records. Under VERSAdos, the resident linker's IDENT command can be used to designate module name, version number, revision number, and description information which will make up the header record. The address field is normally zeros.
- S1 A record containing code/data and the 2-byte address at which the code/data is to reside.
- S2 A record containing code/data and the 3-byte address at which the code/data is to reside.
- S3 A record containing code/data and the 4-byte address at which the code/data is to reside.
- S5 A record containing the number of S1, S2, and S3 records transmitted in a particular block. This count appears in the address field. There is no code/data field.
- S7 A termination record for a block of S3 records. The address field may optionally contain the 4-byte address of the instruction to which control is to be passed. There is no code/data field.
- S8 A termination record for a block of S2 records. The address field may optionally contain the 3-byte address of the instruction to which control is to be passed. There is no code/data field.

- S9 A termination record for a block of S1 records. The address field may optionally contain the 2-byte address of the instruction to which control is to be passed. Under VERSAdos, the resident linker's ENTRY command can be used to specify this address. If not specified, the first entry point specification encountered in the object module input will be used. There is no code/data field.

Only one termination record is used for each block of S-records. S7 and S8 records are usually used only when control is to be passed to a 3- or 4-byte address. Otherwise, an S9 record is used for termination. Normally, only one header record is used, although it is possible for multiple header records to occur.

### CREATION OF S-RECORDS

S-record-format programs may be produced by several dump utilities, debuggers, the VERSAdos resident linkage editor, or several cross assemblers or cross linkers. On VERSAdos systems, the Build Load Module (MBLM) utility allows an executable load module to be built from S-records, and has a counterpart utility in BUILDS, which allows an S-record file to be created from a load module.

Programs are available for downloading or uploading a file in S-record format from a host system to an 8-bit microprocessor-based or a 16-bit microprocessor-based system.

### EXAMPLE

Shown below is a typical S-record-format module, as printed or displayed:

```
S00600004844521B
S1130000285F245F2212226A000424290008237C2A
S11300100002000800082629001853812341001813
S113002041E900084E42234300182342000824A952
S107003000144ED492
S9030000FC
```

The module consists of one S0 record, four S1 records, and an S9 record.

The S0 record is comprised of the following character pairs:

S0 S-record type S0, indicating that it is a header record.

06 Hexadecimal 06 (decimal 6), indicating that six-character pairs (or ASCII bytes) follow.

00 A 4-character, 2-byte address field; zeros in this example.  
00

48  
44 ASCII H, D, and R - "HDR".  
52

1B The checksum.

The first S1 record is explained as follows:

S1 S-record type S1, indicating that it is a code/data record to be loaded/verified at a 2-byte address.

13 Hexadecimal 13 (decimal 19), indicating that 19-character pairs, representing 19 bytes of binary data, follow.

00 A 4-character, 2-byte address field; hexadecimal address 0000; where  
00 the data which follows is to be loaded.

The next 16-character pairs of the first S1 record are the ASCII bytes of the actual program code/data. In this assembly language example, the hexadecimal opcodes of the program are written in sequence in the code/data fields of the S1 records:

<u>OPCODE</u>	<u>INSTRUCTION</u>
285F	MOVE.L (A7)+,A4
245F	MOVE.L (A7)+,A2
2212	MOVE.L (A2),D1
226A0004	MOVE.L 4(A2),A1
24290008	MOVE.L FUNCTION(A1),D2
237C	MOVE.L #FORCEFUNC,FUNCTION(A1)

. (The balance of this code is continued in the  
. code/data fields of the remaining S1 records,  
. and stored in memory location 0010, etc.)

2A The checksum of the first S1 record.

The second and third S1 records each also contain \$13 (19) character pairs and are ended with checksums 13 and 52, respectively. The fourth S1 record contains 07 character pairs and has a checksum of 92.

The S9 record is explained as follows:

- S9 S-record type S9, indicating that it is a termination record.
- 03 Hexadecimal 03, indicating that three character pairs (3 bytes) follow.
- 00 The address field, zeros.
- 00
- FC The checksum of the S9 record.

Each printable character in an S-record is encoded in hexadecimal (ASCII in this example) representation of the binary bits which are actually transmitted. For example, the first S1 record above is sent as:

TYPE				LENGTH				ADDRESS								CODE / DATA								CHECKSUM					
S 1				1 3				0 0 0 0								2 8 5 F ...								2 A					
5 3		3 1		3 1		3 3		3 0		3 0		3 0		3 0		3 2		3 8		3 5		4 6		...		3 2		4 1	
0101	0011	0011	0001	0011	0001	0011	0011	0011	0000	0011	0000	0011	0000	0011	0000	0011	0010	0011	1000	0011	0101	0100	0110	...		0011	0010	0100	0001

**D**

THIS PAGE INTENTIONALLY LEFT BLANK.

## APPENDIX E

### SCSI FIRMWARE SUPPORT

#### E.1 INTRODUCTION

The MVME117 Monoboard Microcomputer contains a port to the Small Computer Systems Interface (SCSI) bus. The hardware interface is realized with a minimum of board space through the NCR 5380 interface chip. A classical firmware/hardware tradeoff was done to include the SCSI bus port on the MVME117. The NCR 5380 has very little intelligence associated with the SCSI bus protocol. This intelligence is provided by the MVME117 SCSI firmware.

To relieve the user of having to follow SCSI bus protocol, the SCSI firmware allows the user to pass commands to the bus through high level command packets. With this method, the firmware interface can greatly speed up the user software development cycle.

#### E.2 MODES OF OPERATION

When using the SCSI firmware, the user has two mode options. Interrupt mode is the most processor-efficient mode of operation. Multitasking is allowed in this mode for targets that support Arbitration, Reselection, and the Message Out Phase. (As far as the MVME117 firmware is concerned, a SASI system is one that does not support Arbitration, Reselection, and the Message Out Phase; while a SCSI system is one that does support Arbitration, Reselection, and the Message Out Phase.) When using the interrupt mode, the user must specify interrupt level 2 in the packet description. (Refer to the packet descriptions in paragraph E.5.) The MC68010 processor is returned to the user whenever the SCSI bus is slowed down (in between phases), or whenever the target disconnects with a pending reselection; this allows commands on the bus to be overlapped.

A slow, processor-inefficient mode called polled mode is also provided for the user who cannot tolerate interrupts. Polled mode is selected by specifying interrupt level 0 in the user packets. This mode provides only a single thread on the SCSI bus. When the user branches to the command entry (refer to paragraph E.3) to perform a command in this mode, the processor stays inside the SCSI firmware until the command is finished or until user interaction is required. Exceptions (e.g., bus parity errors) are checked by polling the registers in the NCR 5380. This checking mechanism is quite slow. Therefore, this non-interrupt polled mode of operation is recommended only for applications that cannot tolerate interrupts.

#### E.3 MVME117 SCSI FIRMWARE ENTRY POINTS

The SCSI firmware resides in two 32K x 8 EPROMs and is co-resident with MVME117BUG, the debug monitor for the MVME117 Monoboard Microcomputer. There are three entry points into the SCSI firmware (entry to the SCSI firmware must be in supervisor state):

- a. Command entry -- the address that is used when a new command is to be executed, i.e., the user points address register A2 to the packet, and branches to this address.
- b. Reactivation entry -- the address used when the continuation of a suspended command is to take place, i.e., the user received an intermediate status of either \$00, \$03, \$04, or \$05 (see below), acted on the particular event, and wants to resume the active command. The user resumes that command by branching to the reactivation entry address. The reactivation entry is also used when the user receives final status with the COMMAND PENDING bit set (see Figure 10).

The following intermediate status codes resume with an interrupt which gives control to the reactivation entry:

- \$00 WAIT TIME: Intermediate status indicating that a time suspension of x milliseconds is requested by the firmware (x is passed in address register A1).
  - \$03 LINK FLAG: Intermediate status indicating that a section of a linked command with flag has been completed. The user may consider this status as a link flag interrupt. (Refer to the SCSI draft revision 14 section 6.2.6.)
  - \$04 MESSAGE RECEIVED: Intermediate status indicating that an extended message has been received that the SCSI firmware does not know how to interpret. The user is allowed to interpret this message and then is expected to branch to the reactivation entry point to resume the particular thread on the SCSI bus.
  - \$05 ALLOCATE MESSAGE BUFFER: The threaded SCSI target has sent an extended message to the MVME117 and the user has not allocated enough buffer space to store the message. The SCSI firmware allows the user to allocate a big enough buffer (the longest possible message is 258 bytes) and is expected to branch back to the reactivation entry to resume the thread on the SCSI bus.
- c. Interrupt entry -- the address of the SCSI firmware interrupt handler. In interrupt mode, the firmware takes over vector number \$44 and points it to the interrupt handler. After an intermediate status of \$01 or \$02 (see below), an NCR 5380 interrupt will vector to the interrupt entry. (An unmaskable, unsolicited interrupt is generated by the NCR 5380 when RST\* is activated. This interrupt also vectors to the interrupt entry.)

The following intermediate status codes resume with an interrupt which gives control to the interrupt entry:

**\$01 WAIT FOR INTERRUPT (CLOSED):** Intermediate status indicating that an NCR 5380 interrupt will bring the processor control back to the SCSI firmware. The MVME117 is currently on the bus and is threaded to the active target that is currently driving BSY\* on the bus. No new commands may be issued to the SCSI firmware until an intermediate status of \$02 or a final status is received.

**\$02 WAIT FOR INTERRUPT (OPEN):** Intermediate status indicating that an NCR 5380 interrupt will bring the processor control back to the SCSI firmware. The MVME117 is currently disconnected from the SCSI bus. (Refer to the interrupt structure in paragraph E.8.) Additional commands may be sent to the SCSI firmware for a different device.

When linked as a part of the MVME117 debug monitor, the SCSI firmware entry points are accessed via the fixed addresses given below:

\$F00048 Command entry.  
\$F0004C Reactivation entry.  
\$F00050 Interrupt entry.

#### E.4 EQUIPMENT SUPPORTED

At this time, the MVME117 1.0 debug monitor supports the following disk controllers:

<u>CONTROLLER</u>	<u>NAME</u>	<u>COMMANDS SUPPORTED</u>	<u>NUMBER</u>
ADSI Saber-AP	Read		08
	Write		0A
	Mode select		15
	Format		04
	Assign alternate sector		07
	Test unit ready		00
	Request sense		03
	Reserve device		16
	Release device		17
DTC - 520 B	Read		08
	Write		0A
	Assign disk parameters		C2
	Define floppy disk track format		C0 sub-opcode 00
	Enable double step function		C0 sub-opcode 01
	Format drive		04
	Assign alternate track		0E
	Test unit ready		00
	Request sense		03

## E.5 EXPLAINING PACKETS

Packets for the following SCSI functions are described in detail:

<u>SCSI FUNCTION NUMBER</u>	<u>DESCRIPTION</u>
\$00	Read
\$04	Write
\$08	Attach
\$0C	Detach
\$10	Format (with/without defect list)
\$14	Assign alternate sector (SCSI)
\$18	Assign alternate track (SASI)
\$1C	Custom SCSI sequence
\$20	SCSI bus reset
\$24	SCSI controller reset

### E

#### E.5.1 Read/Write Packet

Commands sent to controller are: Read (\$08), and Write (\$0A).

For a read or write function, the maximum number of sectors that can be transferred in a single call is 255 (\$FF). If a user needs more, the command must be broken up into several calls.

Figure 1 shows the details of a read/write packet.

	<-----Even Byte-----><-----Odd Byte----->				
	F	C B	8 7	4 3	0
+\$00		Ctlr. LUN		Device LUN	
+\$02		Status Byte 0		Status Byte 1	
+\$04		Memory address MSW			
+\$06		Memory address LSW			
+\$08		Sector Number MSW			
+\$0A		Sector Number LSW			
+\$0C		Number of Sectors to transfer			
+\$0E		0		0	
+\$10		0		0	
+\$12		0		0	
+\$14		0		Function Code	
+\$16		Lvl 0 or 2		Vector Number	
+\$18		Status Byte 2		Status Byte 3	

(Refer to paragraph E.7.)

(Refer to paragraph E.7.)

\$00	00000xxx	Controller Logical Unit Number
\$01	00000xxx	Device Logical Unit Number
\$02	xxxxxxxx	Status from SCSI firmware (Byte 0) (Refer to para-
\$03	xxxxxxxx	Status from SCSI firmware (Byte 1) graph E.7.)
\$04	xxxxxxxx xxxxxxxx	Memory Address (MSW)
\$06	xxxxxxxx xxxxxxxx	Memory Address (LSW)
\$08	xxxxxxxx xxxxxxxx	Sector Number (MSW)
\$0A	xxxxxxxx xxxxxxxx	Sector Number (LSW)
\$0C	00000000 xxxxxxxx	Number of sectors to transfer
\$0E	00000000	Reserved
\$0F	00000000	Reserved
\$10	00000000 00000000	Reserved
\$12	00000000 00000000	Reserved
\$14	00000000	Reserved
\$15	xxxxxxxx	SCSI Function (\$00 = Read, \$04 = Write)
\$16	000000xx	Interrupt level (2 or 0)
\$17	xxxxxxxx	Vector number to use upon return (Usually \$46)
\$18	xxxxxxxx	Status from SCSI firmware (Byte 2) (Refer to para-
\$19	xxxxxxxx	Status from SCSI firmware (Byte 3) graph E.7.)

FIGURE 1. Read/Write Packet

### E.5.2 Attach/Detach Packet

The following SCSI commands are executed during an attach:

<u>DEVICE</u>	<u>COMMAND NAME</u>	<u>CONTROLLER COMMAND</u>	<u>NOTES</u>
SCSI	Reserve device	\$16	If reserve-on-attach bit is set in packet. A reserve device command can be used in a multi-initiator SCSI environment to reserve the device.
	Test unit ready	\$00	
SASI	Assign drive parameters	\$C2	For both Winchester and floppy drives.
	Define floppy media	\$C0 sub-opcode \$00	If attach is for a floppy device.
	Enable/disable double step	\$C0 sub-opcode \$01	If attach is for a floppy device, and media Tracks Per Inch (TPI) is half drive TPI.
	Test unit ready	\$00	

A successful attach is required before any other commands may be sent to the SCSI firmware (except custom SCSI sequence or RESET).

An attach call initializes the SCSI firmware pointers and internal flags, and builds a table of controller/drive attributes. A RAM work area is used by SCSI firmware for building internal pointers and also contains the command sent to the SCSI controller.

Some SCSI controllers return a "check" condition on the first command sent to them after power-up or reset. The SCSI firmware will retry the command if the controller returns "unit attention" request sense information and if the user put a number that was one or greater into the retry field in the attach packet. Otherwise, the request-sense sense data will be returned to the user. The user can then retry the command.

Figure 2 shows the details of an attach/detach packet. Figure 3 details the 160-byte (\$A0-byte) work area specified in the attach packet and used for all subsequent commands. The work area is normally not examined by the user except when the ADDITIONAL STATUS bit is set. (See Figure 10.)

	<-----Even Byte-----><-----Odd Byte----->				
	F	C   B	8   7	4   3	0
+\$00		Ctrlr. LUN			Device LUN
+\$02		Status Byte 0			Status Byte 1
+\$04		0	0		Step Rate
+\$06		Starting Head Number			Number of Heads
+\$08		Number of Cylinders			
+\$0A		Precompensation Cylinder			
+\$0C		Sectors Per Track			
+\$0E		0	0	0	0
+\$10		SCSI Drive Attributes			
+\$12		0	0	0	0
+\$14		0	0		Function Code
+\$16		Lvl 0 or 2			Vector Number
+\$18		Status Byte 2			Status Byte 3
+\$1A		Ctrl. Type	Drive Type		Retry Count
+\$1C		Sector Size			
+\$1E		0	0		# Alt Cyls to reserve
+\$20		RAM Work area Address (MSW)			
+\$22		RAM Work area Address (LSW)			

(Refer to paragraph E.7.)

(Refer to paragraph E.7.)

\$00	00000xxx	Controller Logical Unit Number
\$01	00000xxx	Device Logical Unit Number
\$02	xxxxxxxx	Status from SCSI firmware (Byte 0) (Refer to para-
\$03	xxxxxxxx	Status from SCSI firmware (Byte 1) graph E.7.)
\$04	00000000	Reserved
\$05	xxxxxxxx	Step Rate - Number of 40 us intervals
\$06	xxxxxxxx	Starting Head Number (not used by SCSI)
\$07	xxxxxxxx	Number of heads
\$08	xxxxxxxx xxxxxxxx	Number of Cylinders
\$0A	xxxxxxxx xxxxxxxx	Write Precompensation Cylinder
\$0C	xxxxxxxx xxxxxxxx	Sectors per Track

**FIGURE 2. Attach/Detach Packet (Sheet 1 of 2)**

\$0E	00000000	Reserved	
\$0F	00000000	Reserved	
\$10	xxxxxxxx xxxxxxxx	SCSI Drive Attributes	
	..... 0	FM Encoding 128 Bytes/Sect.	Data Density
	..... 1	MFM Encoding 256 Bytes/Sect.	
	..... 0	Single	Track Density
	..... 1	Double	
	..... 0	Single	Sides
	..... 1	Double	
	..... 0	5-1/4" floppy media	Media Size
	..... 1	8" floppy media	
	..... 0	Floppy Media	Disk Type
	..... 1	Rigid Media	
	..... 0	No Reserve on attach; No Release on Detach	
	..... 1	Reserve on attach; Release on Detach	
	..... 0	Non Buffered Seeks	Seek Type
	..... 1	Buffered Seeks	
	..... 0	Standard IBM format 256b sectors are 128 in TRK 0	
	..... 1	Non Standard All sectors are 256 bytes even TRK 0	
	xxxxxxxx	Reserved	
\$12	00000000 00000000	Reserved	
\$14	00000000	Reserved	
\$15	xxxxxxxx	SCSI Function (\$08 = Attach, \$0C = Detach)	
\$16	000000xx	Interrupt level (2 or 0)	
\$17	xxxxxxxx	Vector number to use upon return (Usually \$46)	
\$18	xxxxxxxx	Status from SCSI firmware (Byte 2) (Refer to para-	
\$19	xxxxxxxx	Status from SCSI firmware (Byte 3) graph E.7.)	
\$1A	xxxx	SCSI Controller Type (4 bits)	
		0-2 = (SASI) Reserved	
		3 = (SASI) DTC 520-B	
		4-7 = (SASI) Reserved	
		8 = (SCSI) Saber-AP	
		9 = (SCSI) Reserved	
		A-F = (SCSI) Reserved	
	xxxx	Drive Type (4 bits)	
	1	Tape	
	0	Direct Access Sequential Device (DASD) (Disk)	
	xxx	Reserved	
\$1B	0000xxxx	Retry Count (\$0F max) - number of SCSI commands	
\$1C	xxxxxxxx xxxxxxxx	Sector Size in bytes	
\$1E	00000000	Reserved	
\$1F	xxxxxxxx	No. of Cylinders to reserve for Alternate Sectors	
\$20	xxxxxxxx xxxxxxxx	Address of 160 byte SCSI RAM Area (MSW)	
\$22	xxxxxxxx xxxxxxxx	SCSI RAM Area (LSW)	

**FIGURE 2. Attach/Detach Packet (Sheet 2 of 2)**

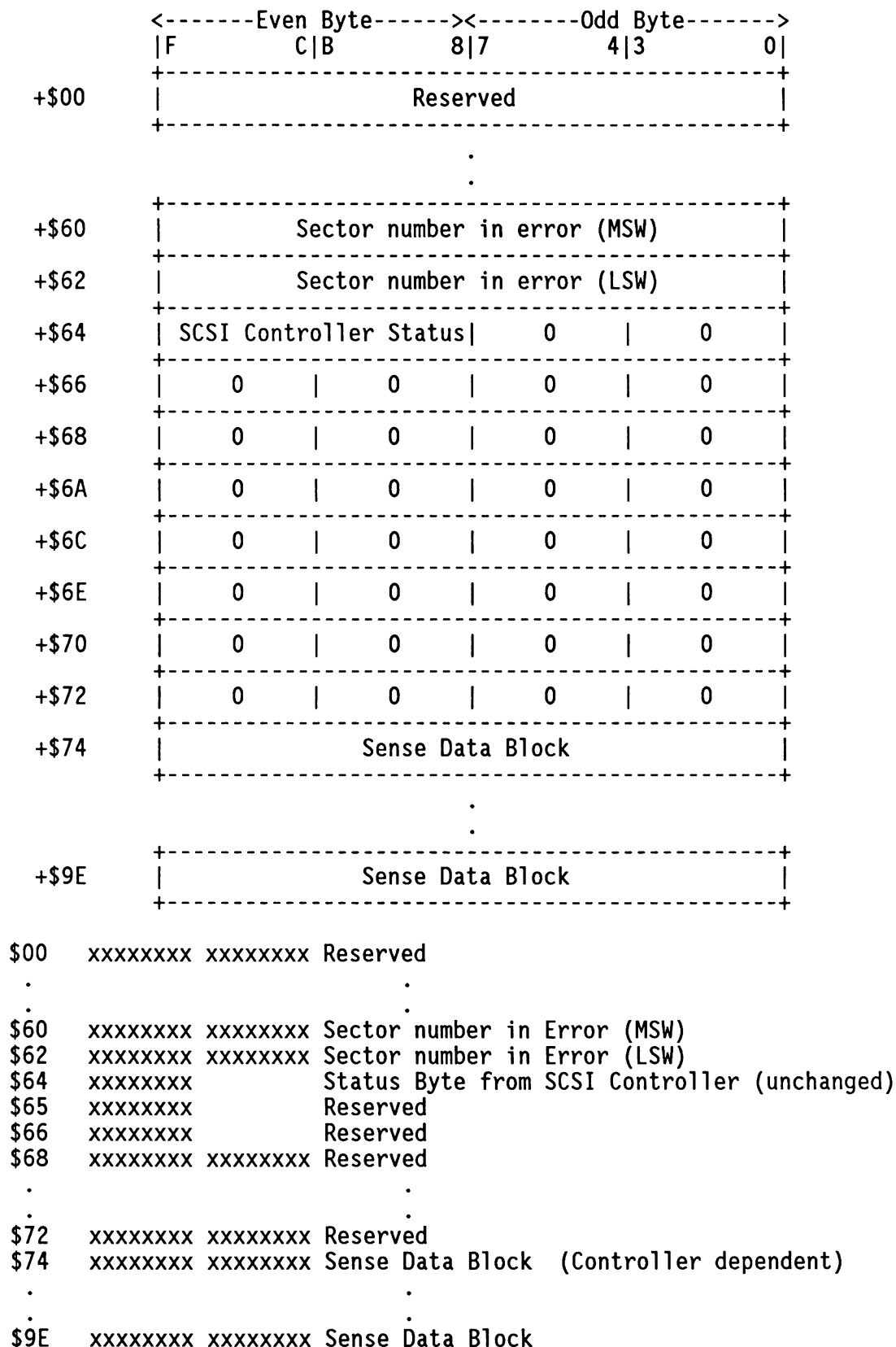


FIGURE 3. 160-Byte Work Area Specified in Attach Packet

### E.5.3 Format Packet

The command sent to the controller is: Format device (\$04). SCSI devices do a mode select (\$15) followed by a format device (\$04); SASI devices do only a format device (\$04).

The defect list is optional for this command. If the number of defects in the defect list is 0, a Format Device command with no defects is sent to the controller. If the field is nonzero, the defect list is sent to the controller. (SASI controllers can not handle defects at format time. They do an Assign Alternate Track command instead.)

Figure 4 shows the details of a format/assign alternate sector packet. Figure 5 is the related defect list.

### E.5.4 Assign Alternate Sector Packet (SCSI)

The SCSI command sent to the controller is: Assign alternate block (\$07).

A size error status is returned if there is not at least one entry in the defect list.

The defect list used is the user's list, but with a byte count added in the second word.

Figure 4 shows the details of a format/assign alternate sector packet. Figure 5 is the related defect list.

	<-----Even Byte-----><-----Odd Byte----->			
	F	C   B	8   7	4   3 0
+\$00		Ctrlr. LUN		Device LUN
+\$02		Status Byte 0		Status Byte 1
+\$04		Number of entries in Defect list		
+\$06		Pointer to Defect list (MSW)		
+\$08		Pointer to Defect list (LSW)		
+\$0A		0	0	0 0
+\$0C		0	0	0 0
+\$0E		Interleave Factor	0	0
+\$10		0	0	0 0
+\$12		0	0	0 0
+\$14		0	0	Function Code
+\$16		Lvl 0 or 2		Vector Number
+\$18		Status Byte 2		Status Byte 3

(Refer to paragraph E.7.)

(Refer to paragraph E.7.)

\$00	00000xxx	Controller Logical Unit Number
\$01	00000xxx	Device Logical Unit Number
\$02	xxxxxxxx	Status from SCSI firmware (Byte 0) (Refer to para-
\$03	xxxxxxxx	Status from SCSI firmware (Byte 1) graph E.7.)
\$04	xxxxxxxx xxxxxxxx	Number of Defects (0 = No Defect List)
\$06	xxxxxxxx xxxxxxxx	Pointer to Defect list (MSW)
\$08	xxxxxxxx xxxxxxxx	Pointer to Defect list (LSW)
\$0A	00000000 00000000	Reserved
\$0C	00000000 00000000	Reserved
\$0E	xxxxxxxx	Interleave Factor
\$0F	00000000	Reserved
\$10	00000000 00000000	Reserved
\$12	00000000 00000000	Reserved
\$14	00000000	Reserved
\$15	xxxxxxxx	SCSI Function (\$10 = Format, \$14 = Assign Alt.)
\$16	000000xx	Interrupt level (2 or 0)
\$17	xxxxxxxx	Vector number to use upon return (Usually \$46)
\$18	xxxxxxxx	Status from SCSI firmware (Byte 2) (Refer to para-
\$19	xxxxxxxx	Status from SCSI firmware (Byte 3) graph E.7.)

FIGURE 4. Format/Assign Alternate Sector Packet

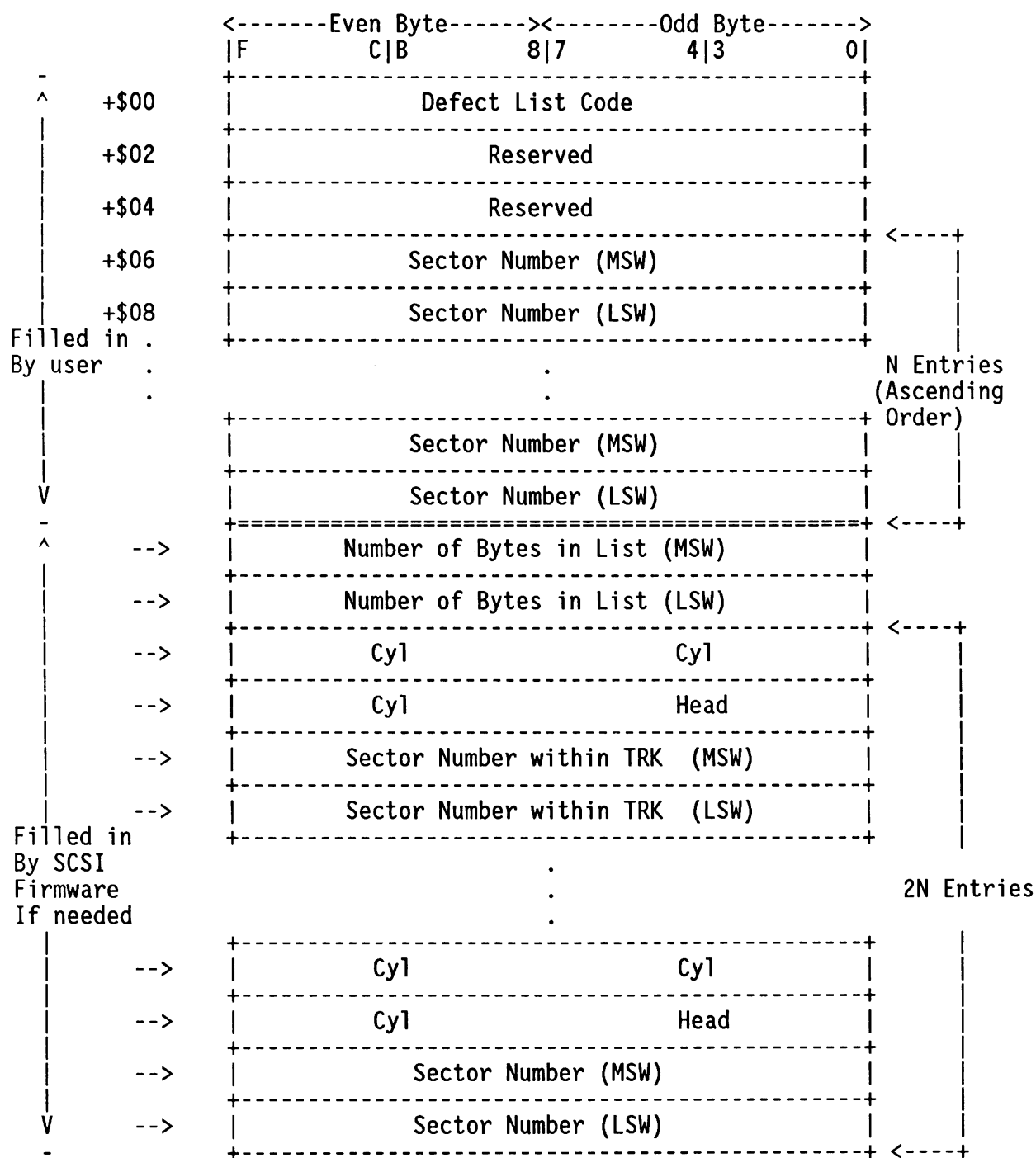


FIGURE 5. Defect List Used by Format and Assign Alternate Sector (SCSI)  
(Sheet 1 of 2)

```

$00  00000000 00000000 Defect List Code
$02  xxxxxxxx xxxxxxxx Reserved
$04  xxxxxxxx xxxxxxxx Reserved
$06  xxxxxxxx xxxxxxxx Sector number (MSW) <-----+
$08  xxxxxxxx xxxxxxxx Sector number (LSW)          |
      .                                           |
      .                                           | N number of bytes
      .                                           |
      .                                           |
      xxxxxxxx xxxxxxxx Sector number (MSW)          |
      xxxxxxxx xxxxxxxx Sector number (LSW) <-----+

```

The packet points to the defect list. The defect list contains defect sector numbers in ascending order. Following the defect list, an "Additional" Work area must be provided. This work area must be equal to  $2N + 1$  longword, (where  $N = \text{Number of Defects}$ ). Some controllers require the defect lists in a different form, and if needed the SCSI firmware will build the list in the area provided. A defect list code of \$0000 implies that the defects are specified as physical sector numbers.

FIGURE 5. Defect List Used by Format and Assign Alternate Sector (SCSI)  
(Sheet 2 of 2)

### E.5.5 Assign Alternate Track Packet (SASI)

The SASI command sent to the controller is: Assign alternate track (\$0E).

The DTC 520 B accepts only one level of reassignment.

Figure 6 shows the details of an assign alternate track packet.

	<-----Even Byte----->				<-----Odd Byte----->			
	F                    C   B		8   7		4   3		0	
+\$00	Ctlr. LUN				Device LUN			
+\$02	Status Byte 0				Status Byte 1			
+\$04	Alternate Sector Number (MSW)							
+\$06	Alternate Sector Number (LSW)							
+\$08	Bad Sector Number (MSW)							
+\$0A	Bad Sector Number (LSW)							
+\$0C	0	0	0	0	0	0	0	
+\$0E	Interleave Factor				0	0	0	0
+\$10	0	0	0	0	0	0	0	
+\$12	0	0	0	0	0	0	0	
+\$14	0	0	Function Code				0	0
+\$16	Lvl 0 or 2				Vector Number			
+\$18	Status Byte 2				Status Byte 3			

(Refer to paragraph E.7.)

(Refer to paragraph E.7.)

(Refer to paragraph E.7.)

(Refer to paragraph E.7.)

\$00	00000xxx	Controller Logical Unit Number
\$01	00000xxx	Device Logical Unit Number
\$02	xxxxxxxx	Status from SCSI firmware (Byte 0) (Refer to para-
\$03	xxxxxxxx	Status from SCSI firmware (Byte 1) graph E.7.)
\$04	xxxxxxxx xxxxxxxx	Alternate Sector Number (MSW)
\$06	xxxxxxxx xxxxxxxx	Alternate Sector Number (LSW)
\$08	xxxxxxxx xxxxxxxx	Bad Sector Number (MSW)
\$0A	xxxxxxxx xxxxxxxx	Bad Sector Number (LSW)
\$0C	00000000 00000000	Reserved
\$0E	xxxxxxxx	Interleave Factor
\$0F	00000000	Reserved
\$10	00000000 00000000	Reserved
\$12	00000000 00000000	Reserved
\$14	00000000	Reserved
\$15	xxxxxxxx	SCSI Function (\$18 = Assign Alt. Track - SASI)
\$16	000000xx	Interrupt level (2 or 0)
\$17	xxxxxxxx	Vector number to use upon return (Usually \$46)
\$18	xxxxxxxx	Status from SCSI firmware (Byte 2) (Refer to para-
\$19	xxxxxxxx	Status from SCSI firmware (Byte 3) graph E.7.)

FIGURE 6. Assign Alternate Track Packet (SASI)

### E.5.6 Custom SCSI Sequence Packet

This function lets the user interface to devices not specifically supported by the MVME117 1.x firmware. (Paragraph E.4 lists devices that are supported.) The function enables a user to do any SASI/SCSI sequence with a SASI/SCSI target. The user must build the custom SCSI sequence packet, build the command table, and the SCSI script. A custom SCSI sequence might be:

```
. build packet \
. build command table |
. point to script > Example follows.
. build message-out |
. build message-in /
. point return vector to return handler
. point register A2 to packet
. jump to command entry \
    . intermediate return: do RTS or RTE | Refer to paragraph E.9,
    . final return: look at status | Writing a Driver.
```

Example: Packet

Address	Data	Description
\$010000	\$0301	Controller level = 3, device LUN = 1
\$010002	\$0000	Returned status bytes 0 and 1
\$010004	\$0001	Script pointer (\$11000) MSW
\$010006	\$1000	Script pointer LSW
\$010008	\$0001	Command Table pointer (\$11200) MSW
\$01000A	\$1200	Command Table pointer LSW
\$01000C	\$0000	Reserved
\$01000E	\$0000	Initiator role
\$010010	\$0000	Reserved
\$010012	\$0000	Reserved
\$010014	\$001C	Function code
\$010016	\$0246	Interrupt level = 2, return vector number = \$46
\$010018	\$0000	Returned status bytes 2 and 3
\$01001A	\$0003	Retry count = 3

### Example: Command Table

<u>Address</u>	<u>Data</u>	<u>Description</u>
\$011200	\$7003	Parity enabled, DMA off, CSCSI = 1, SCSI mode, retry count = 3
\$011202	\$0000	Link pointer MSW
\$011204	\$0000	Link pointer LSW
\$011206	\$000A	Command length = 10 bytes
\$011208	\$2802	SCSI command = \$28 (Extended Read), LUN = 1
\$01120A	\$0000	Sector number (\$64) MSW
\$01120C	\$0064	Sector number LSW
\$01120E	\$0000	Reserved, number of sectors (1) MSB
\$011210	\$0100	Number of sectors LSB, reserved
\$011212	\$0000	Reserved
\$011214	\$0000	SCSI status
\$011216	\$0000	Data length (\$100) MSW
\$011218	\$0100	Data length LSW
\$01121A	\$0002	Data pointer (\$20000) MSW
\$01121C	\$0000	Data pointer LSW
\$01121E	\$0001	Message in length (1)
\$011220	\$0001	Message in pointer (\$13000) MSW
\$011222	\$3000	Message in pointer LSW
\$011224	\$0001	Message out length (1)
\$011226	\$0001	Message out pointer (\$14000) MSW
\$011228	\$4000	Message out pointer LSW

### Example: Script

<u>Address</u>	<u>Data</u>	<u>Description</u>
\$011000	\$14	Message-out phase
\$011001	\$04	Command phase
\$011002	\$0C	Data-in phase
\$011003	\$10	Status phase
\$011004	\$18	Message-in phase
\$011005	\$00	Script done

### Example: Message-in Buffer

<u>Address</u>	<u>Data</u>	<u>Description</u>
\$013000	\$XX	Message-in area

### Example: Message-out Buffer

<u>Address</u>	<u>Data</u>	<u>Description</u>
\$014000	\$C1	Message-out area (identify LUN 1 message)

Figure 7 shows the details of a custom SCSI sequence packet. Paragraph E.6 gives further details of the sequence, including command table and script.

	<-----Even Byte-----><-----Odd Byte----->				
	F	C B	8 7	4 3	0
+\$00		Ctlr. LUN		Device LUN	
+\$02		Status Byte 0		Status Byte 1	
+\$04		Script Pointer (MSW)			
+\$06		Script Pointer (LSW)			
+\$08		Command Table Pointer (MSW)			
+\$0A		Command Table Pointer (LSW)			
+\$0C		0		0	
+\$0E		Target/Init		0	
+\$10		0		0	
+\$12		0		0	
+\$14		0		Function Code	
+\$16		Lvl 0 or 2		Vector Number	
+\$18		Status Byte 2		Status Byte 3	
+\$1A		0		Retry Count	

(Refer to paragraph E.7.)

(Refer to paragraph E.7.)

E

FIGURE 7. Custom SCSI Sequence Packet (Sheet 1 of 2)

\$00	00000xxx	Controller Logical Unit Number
\$01	00000xxx	Device Logical Unit Number
\$02	xxxxxxxx	Status from SCSI firmware (Byte 0) (Refer to para-
\$03	xxxxxxxx	Status from SCSI firmware (Byte 1) graph E.7.)
\$04	xxxxxxxx xxxxxxxx	Script Pointer (MSW)
\$06	xxxxxxxx xxxxxxxx	Script Pointer (LSW)
\$08	xxxxxxxx xxxxxxxx	Command Table Pointer (MSW)
\$0A	xxxxxxxx xxxxxxxx	Command Table Pointer (LSW)
\$0C	00000000 00000000	Reserved
\$0E	1.....	Target Role
	0.....	Initiator Role (Target Enable/Seq. bit undefined)
	.1.....	Target Enable
	.0.....	Target Sequence
	..000000	Reserved
\$0F	00000000	Reserved
\$10	00000000 00000000	Reserved
\$12	00000000 00000000	Reserved
\$14	00000000	Reserved
\$15	xxxxxxxx	SCSI Function (\$1C = Custom SCSI Sequence)
\$16	000000xx	Interrupt level (2 or 0)
\$17	xxxxxxxx	Vector number to use upon return (Usually \$46)
\$18	xxxxxxxx	Status from SCSI firmware (Byte 2) (Refer to para-
\$19	xxxxxxxx	Status from SCSI firmware (Byte 3) graph E.7.)
\$1A	00000000	Reserved
\$1B	0000xxxx	Retry Count (\$0F max) - number of SCSI commands

NOTE: For Initiator... packet is to be provided as described above.  
For Target Enable... no Script pointer or Controller LUN.  
For Target Sequence... Device LUN = 0, and upon return the  
packet will reflect the Initiator's SCSI Level.

FIGURE 7. Custom SCSI Sequence Packet (Sheet 2 of 2)

### E.5.7 SCSI Bus Reset Packet

This function activates the RST\* (RESET) line on the SCSI bus (hardware reset). All devices on the bus are reset and any commands that were pending are lost.

The SCSI firmware cleans up some internal flags and builds a table (see below) of all outstanding packet pointers. The status returned to the user is "command complete" (\$00), and register A0 points to the table of outstanding packets. The last entry is zero, indicating the end of the table.

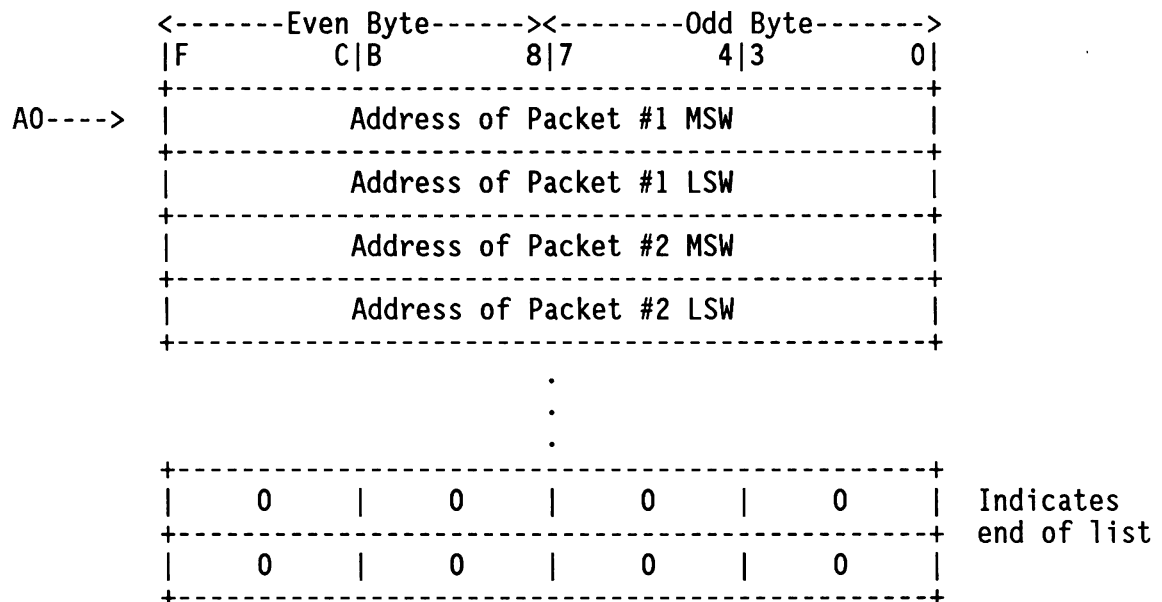


Figure 8 shows the details of a SCSI bus reset/SCSI controller reset packet.

### E.5.8 SCSI Controller Reset Packet

This function sends a bus device reset (\$0C) message to the controller specified in the packet. A table (see above) of outstanding packet pointers within this controller is returned (pointed to by register A0) along with the status in the packet. This command affects only the specified controller.

Figure 8 shows the details of a SCSI bus reset/SCSI controller reset packet.

	<-----Even Byte-----><-----Odd Byte----->				
	F	C B	8 7	4 3	0
+\$00		Ctrlr. LUN (opt)		0	0
+\$02		Status Byte 0		Status Byte 1	
+\$04		0	0	0	0
+\$06		0	0	0	0
+\$08		0	0	0	0
+\$0A		0	0	0	0
+\$0C		0	0	0	0
+\$0E		0	0	0	0
+\$10		0	0	0	0
+\$12		0	0	0	0
+\$14		0	0	Function Code	
+\$16		Lvl 0 or 2		Vector Number	
+\$18		Status Byte 2		Status Byte 3	

(Refer to paragraph E.7.)

(Refer to paragraph E.7.)

\$00	00000000	Controller Logical Unit Number (Ctrlr. Reset only)
\$01	00000000	Reserved
\$02	xxxxxxxx	Status from SCSI firmware (Byte 0) (Refer to para-
\$03	xxxxxxxx	Status from SCSI firmware (Byte 1) graph E.7.)
\$04	00000000 00000000	Reserved
\$06	00000000 00000000	Reserved
\$08	00000000 00000000	Reserved
\$0A	00000000 00000000	Reserved
\$0C	00000000 00000000	Reserved
\$0E	00000000 00000000	Reserved
\$10	00000000 00000000	Reserved
\$12	00000000 00000000	Reserved
\$14	00000000	Reserved
\$15	xxxxxxxx	SCSI Function (\$20=Bus Reset) (\$24=Ctrlr. Reset)
\$16	000000xx	Interrupt level (2 or 0)
\$17	xxxxxxxx	Vector number to use upon return (Usually \$46)
\$18	xxxxxxxx	Status from SCSI firmware (Byte 2) (Refer to para-
\$19	xxxxxxxx	Status from SCSI firmware (Byte 3) graph E.7.)

FIGURE 8. SCSI Bus Reset/SCSI Controller Reset Packet

## E.6 CUSTOM SCSI SEQUENCE DETAILS

The MVME117 1.x SCSI firmware is based on the SCSI bus phases. These phases are as given in Table 1.

TABLE 1. SCSI Bus Phases

PHASE	DIRECTION	NOTES
Bus Free		No activity on the bus. SEL* and BSY* are not activated.
Arbitration		SCSI devices arbitrate for the use of the bus by activating BSY* and their ID.
(Re)Selection		One SCSI device selects another device by activating SEL* along with its ID and the ID for the other device.
Information Transfer Phases:		
Command	initiator to target	A command tells the target what is requested by the initiator. The Command Descriptor Block (CDB) is passed during this phase.
Status	target to initiator	The status of a particular command is passed to initiator. Examples: good, busy, check.
Data in	target to initiator	Data is transferred from the target to the initiator as a result of a data phase requested in the CDB.
Data out	initiator to target	Data is transferred from the initiator to the target as a result of a data phase requested in the CDB.
Message in	target to initiator	Messages are sent to the initiator to send bus, command, and controller information. Examples: command complete, save data pointer, restore data pointer, message reject.
Message out	initiator to target	Messages are sent to the target to send bus, command, and controller information. Examples: identify, initiator detected error, abort, device reset.

E

In general, the SCSI information transfer phases are controlled by the target that is presently driving BSY\* on the bus. The initiator follows the phases that the target wants to perform.

The MVME117 SCSI firmware follows the information transfer phase sequences by means of a list called a "script". ("Script" will be used for this list from now on.) The script is the list of expected phases which must be completed to execute a command over the SCSI bus.

The following example of a script is for a read command (CDB = Read):

MESSAGEOUT	Message out phase: initiator sends an identify message telling the target that reselection is supported. The particular device LUN is encoded in the message.
COMMAND	Command phase: a read command is encoded in the six bytes that are transferred in the command phase to tell which block to transfer and how many blocks to transfer.
DATAIN	Data in phase: data is transferred from the target to the initiator. The number of bytes transferred equals the number of blocks desired times the block size.
STATUS	Status phase: a one-byte status is transferred from the target to the initiator to indicate that the transfer from the media to the host was good.
MESSAGEIN	Message in: the target sends a one-byte message to the controller to indicate that the command is complete.
LISTDONE	Done: end of script.

The script pointer is passed to the MVME117 SCSI firmware through the custom SCSI sequence packet. (Refer to paragraph E.5.6.) The script codes are:

TABLE 2. Custom SCSI Script Codes

CODE	NAME	MEANING
\$00	LISTDONE	End of script
\$04	COMMAND	Command phase
\$08	DATAOUT	Data out phase
\$0C	DATAIN	Data in phase
\$10	STATUS	Status phase
\$14	MESSAGEOUT	Message out phase
\$18	MESSAGEIN	Message in phase

The SCSI firmware uses the script and also a table of information. This table is called the Command Table (CT). The CT pointer is passed to the SCSI firmware through the custom SCSI sequence packet. The CT is detailed in Figure 9.

	<-----Even Byte-----><-----Odd Byte----->			
	F	C B	8 7	4 3 0
+\$00	Status/Flag Byte		0	Retry Count
+\$02	Link Pointer (MSW)			
+\$04	Link Pointer (LSW)			
+\$06	Command Length			
+\$08	SCSI Command Descriptor Block (CDB)			
+\$0A	SCSI Command Descriptor Block (CDB)			
+\$0C	SCSI Command Descriptor Block (CDB)			
+\$0E	SCSI Command Descriptor Block (CDB)			
+\$10	SCSI Command Descriptor Block (CDB)			
+\$12	SCSI Command Descriptor Block (CDB)			
+\$14	SCSI Status		0	0
+\$16	Data Length (MSW)			
+\$18	Data Length (LSW)			
+\$1A	Data Pointer (MSW)			
+\$1C	Data Pointer (LSW)			
+\$1E	Message In Length			
+\$20	Message In Pointer (MSW)			
+\$22	Message In Pointer (LSW)			
+\$24	Message Out Length			
+\$26	Message Out Pointer (MSW)			
+\$28	Message Out Pointer (LSW)			

\$00	xxxxxxx	Status/Flag Byte
	0.....	Lnk--Link Flag Bit disabled
	1.....	Lnk--link Command Tables, support linked commands
	.0.....	Parity disabled
	.1.....	Parity enabled--MVME117 checks SCSI bus parity
	..0.....	DMA on

FIGURE 9. Command Table (CT) (Sheet 1 of 2)

	...1.....	DMA Off Flag--disable DMA for data out/in (NOTE 1)
	...0.....	CSCSI--Custom Sequence Flag:checks status (NOTE 2)
	...1.....	CSCSI--does not check SCSI status (NOTE 2)
	....0....	SCSI firmware uses SCSI rules (NOTE 3)
	....1....	SASI Mode Flag--firmware uses SASI rules (NOTE 3)
	.....xxx	Reserved
\$01	0000xxxx	Retry Count (\$0F max)--No. times retried (NOTE 4)
\$02	xxxxxxxx xxxxxxxx	Link Pointer (MSW)--For linked commands. Valid
\$04	xxxxxxxx xxxxxxxx	Link Pointer (LSW) only if Link Flag Bit = 1.
\$06	xxxxxxxx xxxxxxxx	Command Length--Length of the CDB (in bytes)
\$08	xxxxxxxx xxxxxxxx	SCSI Command Descriptor Block (CDB)--SCSI draft
\$0A	xxxxxxxx xxxxxxxx	SCSI Command Descriptor Block (CDB) rev. 14
\$0C	xxxxxxxx xxxxxxxx	SCSI Command Descriptor Block (CDB) allows
\$0E	xxxxxxxx xxxxxxxx	SCSI Command Descriptor Block (CDB) 12 bytes
\$10	xxxxxxxx xxxxxxxx	SCSI Command Descriptor Block (CDB) maximum
\$12	xxxxxxxx xxxxxxxx	SCSI Command Descriptor Block (CDB) length
\$14	xxxxxxxx	SCSI Status
\$15	00000000	Reserved
\$16	xxxxxxxx xxxxxxxx	Data Length (MSW)--No. bytes expected during
\$18	xxxxxxxx xxxxxxxx	Data Length (LSW) data in or data out phase
\$1A	xxxxxxxx xxxxxxxx	Data Pointer (MSW)--To memory area where firmware reads (data out) or writes
\$1C	xxxxxxxx xxxxxxxx	Data Pointer (LSW) (data in) (contiguous buffer)
\$1E	xxxxxxxx xxxxxxxx	Msg. In Length--Bytes expected during message in (max=258 for extended messages)
\$20	xxxxxxxx xxxxxxxx	Msg. In Pointer (MSW)--to RAM buffer where firm-
\$22	xxxxxxxx xxxxxxxx	Msg. In Pointer (LSW) ware stores recv'd msgs.
\$24	xxxxxxxx xxxxxxxx	Msg. Out Length-- Bytes expected to be transfer- red in message out phase (max=258 for extended messages)
\$26	xxxxxxxx xxxxxxxx	Msg. Out Pointer(MSW)--to RAM buffer where firm-
\$28	xxxxxxxx xxxxxxxx	Msg. Out Pointer(LSW) ware takes msgs. to transfer to target

**NOTES:**

- (1) If = 1, do not use fast pseudo-DMA channel for data out/in phases.

**CAUTION**

**TARGET MUST MAINTAIN CONTINUOUS DATA RATE FOR THIS TO BE USED RELIABLY.**

- (2) If = 0, SCSI firmware interprets returned SCSI status, and sends a request sense command to the controller if status is "check". If = 1, SCSI firmware does not read the SCSI status from the CT, and returned status word in the packet reflects only firmware status.
- (3) If = 0, SCSI firmware first arbitrates for the bus, sends an identify message, and supports disconnection and reselection. If = 1, SCSI firmware operates by SASI rules (no bus arbitration, no message-out identify message, and no disconnection or reselection).
- (4) SCSI bus parity errors are retried this many times, if needed.

FIGURE 9. Command Table (CT) (Sheet 2 of 2)

## E

15	08    07	00
<b>C O N T R O L   F L A G S</b>		<b>S T A T U S   C O D E</b> (REFER TO TABLE 3)
<div style="margin-left: 40px;">+-----Bits 11 - 8 reserved</div> <div style="margin-left: 20px;">+-----Bit 12 (COMMAND PENDING) 1 = a command was sent to the firmware and reactivation is required to resume. The firmware has pre- processed and queued the command. (Refer to para- graph E.9.4.1.) 0 = no reactivation requested.</div> <div style="margin-left: 20px;">+-----Bit 13 (RTE FLAG) 1 = this return was <u>not</u> preceded by an interrupt and is the first return since command entry. In this case, no RTE is required. 0 = this return was preceded by an interrupt and is not the first return, therefore, an RTE is required to continue processing from where an interrupt occurred. Register A3 has a pointer to a register save area (D0 - D7, A0 - A6).</div> <div style="margin-left: 20px;">+-----Bit 14 (ADDITIONAL STATUS) 1 = external status in STATAREA (CT + \$96). 0 = external status is not valid.</div> <div style="margin-left: 20px;">+-----Bit 15 (FINALSTAT) 1 = intermediate return. 0 = final status. The script processing completed successfully, OR the script processing encountered a fatal error.</div>		

This does not mean that the operation that the user requested on the SCSI was successful. The status is contained in the status code (bits 7 - 0).

181

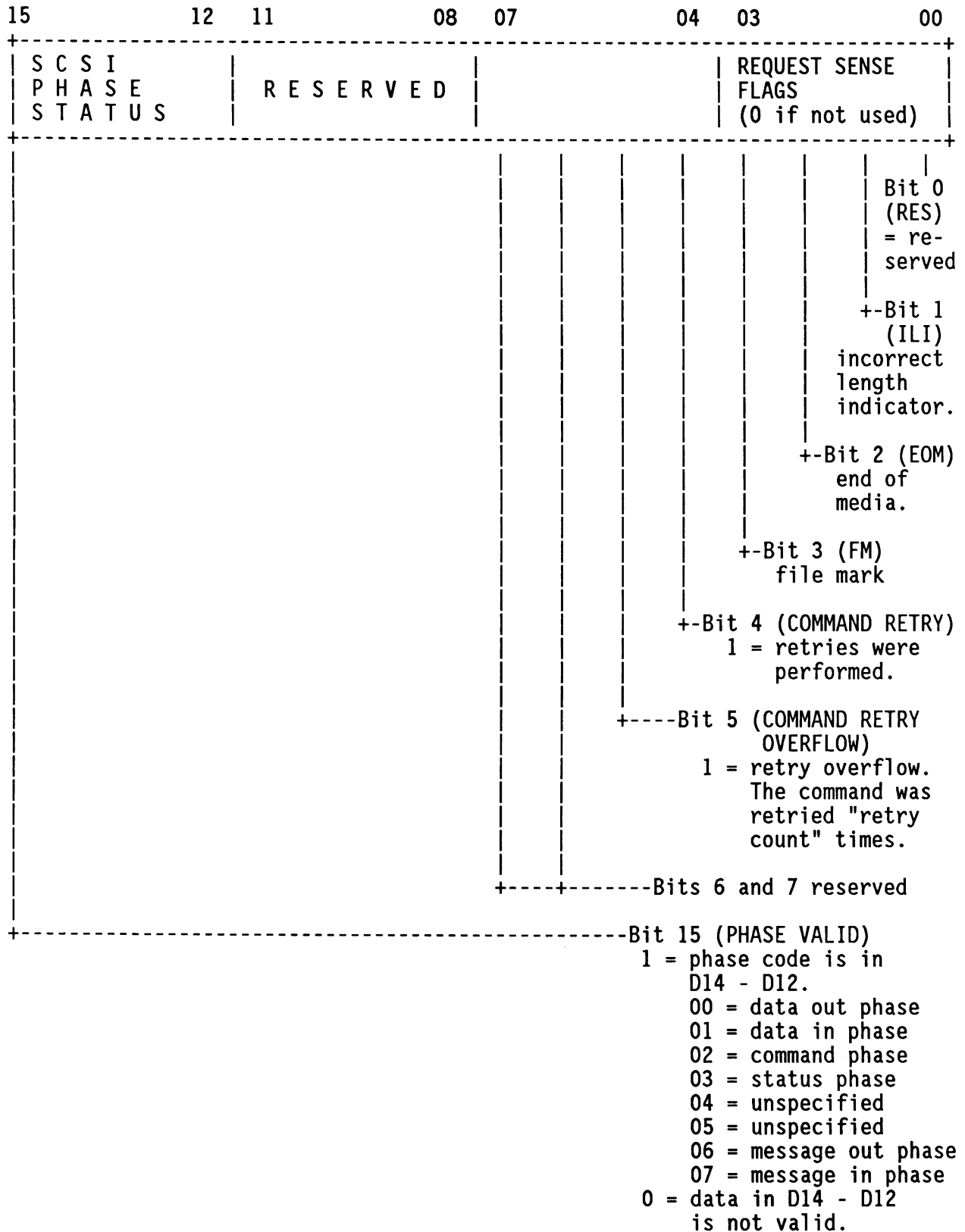


FIGURE 11. Second Status Word (Offset of \$18 Bytes in Packet)

TABLE 3. Packet Status Codes

CODE	MEANING	NOTES
<u>INTERMEDIATE RETURN CODES</u>		
\$00	Wait time; A1 contains the MINIMUM delay time in msec.	(1)
\$01	Wait for interrupt; command door closed. No new commands may be issued to firmware.	(1)
\$02	Wait for interrupt; command door open. OK to send new commands for other devices to firmware.	(1)
\$03	Link flag received.	(1)
\$04	A message has been received. User must interpret.	(1)
\$05	Allocate message buffer; an extended message has been received.	(1)
<u>FINAL RETURN CODES</u>		
\$00	GOOD. Script processing is OK.	(2)
\$01	Undefined problem.	(2)
\$02	REACTIVATE was entered with no reactivate request pending.	(2)
\$03	Interrupt handler was entered with no pending IRQ.	(2)
\$04	Reselection not expected from this target.	(2)
\$05	Target thinks it is working on linked commands but the Command Table does not.	(2)
\$06	Linked command has error status code; Command has been aborted.	(2)
\$07	Received an illegal message.	(2)
\$08	The message we have tried to send was rejected.	(2)
\$09	Encountered a parity error in data-in phase, command phase (target only), status phase, or message-in phase. (Refer to bits 15 - 12 of second status word. See Figure 11.)	(2)
\$0A	SCSI bus reset received (A1 pointing to packet list).	(2)
\$0B	Command error (bad command code, bad timing, or command door was closed when a command was received).	(2)
\$0C	Size error (number of sectors too big, invalid format code).	(2)
\$0D	Bad ID in packet or local ID (\$F43FC7).	(2)
\$0E	Error in attach (not previously attached, bad device LUN, unsupported controller).	(2)
\$0F	Busy error (device has a command pending).	(2)
\$10	There is disagreement between initiator and target regarding the number of bytes that are to be transferred. If bit 15 of status = 1, then bits 12 - 14 contain the phase code.	(2)
\$11	Received a BERR* while in DMA mode from a device that did not respond fast enough. The controller must be capable of moving data at least 10 kilobytes per second in DMA mode.	(2)
\$12	Selection time-out. Target does not respond.	(2)
\$13	SCSI protocol violation. SASI mode and bus was busy.	(2)
\$14	Script mismatch. CHECK STATUS. If SCSI status within Command Table (offset \$14 for Custom Sequence, otherwise \$64) is zero, then assume Script mismatch, otherwise use SCSI packet status.	(2)
\$15	Script mismatch. The target sequence of operation did not match the script.	(2)
\$16	Illegal SCSI state machine transition.	(2)
\$17	Command has been received (in target role).	(2)
\$18	Script complete in target role.	(2)
\$19	Script complete and new command loaded (target role).	(2)
\$1A	Target module called. Target role not supported.	(2)
\$1B	Reserved and unused.	(2)

TABLE 3. Packet Status Codes (cont'd)

CODE	MEANING	NOTES
\$1C	Reserved and unused.	(2)
\$1D	Reserved and unused.	(2)
\$1E	Reserved and unused.	(2)
\$1F	Reserved.	(2)
<u>Request-Sense-Data Error-Class 7 Codes (Controller Dependent)</u>		
\$20	NO SENSE. Indicates that there is no specific sense key information to be reported for the designated logical unit.	(2,3)
\$21	RECOVERED ERROR. Indicates that the last command completed successfully with some recovery action performed by the target. Details can be determined by examining the additional sense bytes and information bytes.	(2,3)
\$22	NOT READY. Indicates that the logical unit addressed cannot be accessed. Operator intervention may be required to correct this condition.	(2,3)
\$23	MEDIUM ERROR. Indicates that the target detected a non-recoverable error condition that was probably caused by a flaw in the medium or an error in recording data.	(2,3)
\$24	HARDWARE ERROR. Indicates that the target detected a non-recoverable hardware failure (for example, controller failure, device failure, parity error, etc.) while performing the command or during self test.	(2,3)
\$25	ILLEGAL REQUEST. Indicates that there was an illegal parameter in the command descriptor block or in the additional parameters supplied as data.	(2,3)
\$26	UNIT ATTENTION. Indicates that the removeable media may have been changed or the target has been reset.	(2,3)
\$27	DATA PROTECT. Indicates that a command that reads or writes the medium was attempted on a block that is protected from this operation.	(2,3)
\$28	BLANK CHECK. Indicates that a write-once read-multiple device or a sequential access device encountered a blank block while reading or a write-once read-multiple device encountered a nonblank block while writing.	(2,3)
\$29	VENDOR UNIQUE. Used for reporting vendor unique conditions (for Saber AP = Format Complete).	(2,3)
\$2A	COPY ABORTED. Indicates that a Copy or a Copy and Verify command was aborted due to an error condition.	(2,3)
\$2B	ABORTED COMMAND. Indicates that the target aborted the command. The initiator may be able to recover by trying the command again.	(2,3)
\$2C	EQUAL. Indicates a Search Data command has satisfied an equal comparison.	(2,3)
\$2D	VOLUME OVERFLOW. Indicates that a buffered peripheral device has reached an end-of-medium and data remains in the buffer that has not been written to the medium. A Recover Buffered Data command may be issued to read the unwritten data from the buffer.	(2,3)
\$2E	MISCOMPARE. Indicates that the source data did not match the data read from the medium.	(2,3)
\$2F	RESERVED. This sense key is reserved.	(2,3)

TABLE 3. Packet Status Codes (cont'd)

CODE	MEANING	NOTES
<u>SCSI Status Returned in Status Phase</u>		
\$31	SCSI status = \$02. CHECK.	(2,4)
\$32	SCSI status = \$04. CONDITION MET.	(2,4)
\$34	SCSI status = \$08. BUSY.	(2,4)
\$38	SCSI status = \$10. INTERMEDIATE / GOOD.	(2,4)
\$3A	SCSI status = \$14. INTERMEDIATE / CONDITION MET / GOOD.	(2,4)
\$3C	SCSI status = \$18. RESERVATION CONFLICT.	(2,4)
<u>Request-Sense-Data Error-Classes 0 - 6 Codes (Controller Dependent)</u>		
\$40	NO ERROR STATUS.	(2,5,6)
\$41	NO INDEX SIGNAL.	(2,5,6)
\$42	NO SEEK COMPLETE.	(2,5,6)
\$43	WRITE FAULT.	(2,5,6)
\$44	DRIVE NOT READY.	(2,5,6)
\$45	DRIVE NOT SELECTED.	(2,5,6)
\$46	NO TRACK 00.	(2,5,6)
\$47	MULTIPLE DRIVES SELECTED.	(2,5,6)
\$49	CARTRIDGE CHANGED.	(2,5,6)
\$4D	SEEK IN PROGRESS.	(2,5,6)
\$50	ID ERROR. ECC error in the data field.	(2,5,7)
\$51	DATA ERROR. Uncorrectable data error during a read.	(2,5,7)
\$52	ID ADDRESS MARK NOT FOUND.	(2,5,7)
\$53	DATA ADDRESS MARK NOT FOUND.	(2,5,7)
\$54	SECTOR NUMBER NOT FOUND.	(2,5,7)
\$55	SEEK ERROR.	(2,5,7)
\$57	WRITE PROTECTED.	(2,5,7)
\$58	CORRECTABLE DATA FIELD ERROR.	(2,5,7)
\$59	BAD BLOCK FOUND.	(2,5,7)
\$5A	FORMAT ERROR. (Check track command.)	(2,5,7)
\$5C	UNABLE TO READ ALTERNATE TRACK ADDRESS.	(2,5,7)
\$5E	ATTEMPTED TO DIRECTLY ACCESS AN ALTERNATE TRACK.	(2,5,7)
\$5F	SEQUENCER TIME OUT DURING TRANSFER.	(2,5,7)
\$60	INVALID COMMAND.	(2,5,8)
\$61	ILLEGAL DISK ADDRESS.	(2,5,8)
\$62	ILLEGAL FUNCTION.	(2,5,8)
\$63	VOLUME OVERFLOW.	(2,5,8)
\$70	RAM ERROR. (DTC 520-B)	(2,5,9)
\$71	FDC 765 ERROR. (DTC 520-B)	(2,5,9)

## NOTES:

- (1) Intermediate return codes. Bit 15=1, actual word=\$80xx, \$90xx, etc.
- (2) Final return codes.
- (3) Sense key status codes for request-sense-data error-class 7. An offset of \$20 is added to all sense key codes.
- (4) The SCSI status sent from the controller is ANDed with \$1E, shifted right one bit, and \$30 added.
- (5) Sense key status codes for request-sense-data error-classes 0 - 6. An offset of \$40 is added to all sense key codes.
- (6) Drive error codes.
- (7) Controller error codes.
- (8) Command errors.
- (9) Miscellaneous errors.

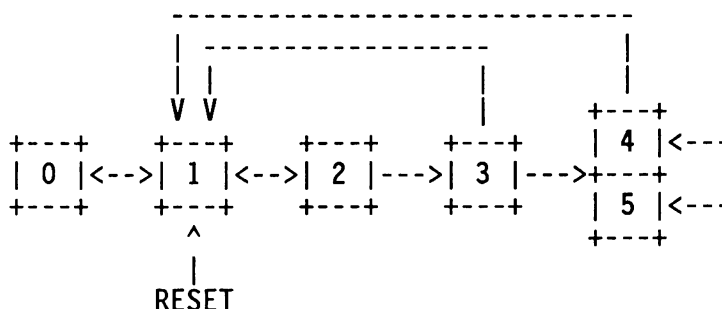
## E.8 SCSI FIRMWARE INTERRUPT STRUCTURE

The MVME117 SCSI firmware is based on the NCR 5380 SCSI bus interface chip. Seven programmable interrupts are used to signal the processor when a significant event takes place.

<u>INTERRUPT SOURCE</u>	<u>INTERRUPT TYPE</u>
1	Bus phase mismatch
2	Loss-of-busy
3	(Re)Selection
4	Bus parity error
5	(Re)Selection parity error
6	Bus reset
7	EOP = DMA End-Of-Process (not implemented on MVME117)

A state machine is implemented in the firmware to transition through the four states or "phases" of the SCSI bus. The SCSI firmware uses two extra states for a total of six states that somewhat follow the four phases of the SCSI bus. The extra DMA state (state 5) is no different from the non-DMA state (state 4), and is used for diagnosing problems (see below). The extra idle/ready state (state 0) is different from the bus free state (state 1) because the firmware may be idle while the SCSI bus is not.

<u>STATE NUMBER</u>	<u>DESCRIPTION</u>	<u>SELECTION</u>
State 0	Idle/ready (disconnected from the bus)	Forced by enabling the (Re)Selection interrupt.
State 1	Bus free state (transitional)	Detected by monitoring the Loss-of-busy interrupt.
State 2	Arbitration	Only detected by polling one of the bits in control register #2 of the NCR 5380. No interrupt is generated for this event.
State 3	(Re)Selection	Detected by monitoring the (Re)Selection interrupt.
State 4	Information transfer (without DMA)	Detected by monitoring the Bus phase mismatch interrupt.
State 5	Information transfer (with DMA)	Detected by monitoring the Bus phase mismatch interrupt.



The MVME117 SCSI interface firmware is designed for processor efficiency. Whenever the SCSI bus is in a state that does not need monitoring, the firmware releases the processor so it may perform other functions such as user tasks and lower priority events. In these cases, an interrupt brings processor control back to the firmware.

A return vector is provided to the SCSI firmware in all cases through the packet pointed to by register A2. Whenever the firmware returns to the user through this return vector, it flags whether the processor was brought back to the firmware through an external interrupt. This flagging is done by the RTE FLAG bit in the status word stored in the user packet. (See Figure 10.) If the bit = 1, no RTE is to be performed by the user. If the bit = 0, eventually an RTE is required by the user to return the processor to the interrupted task.

Similarly, whenever the RTE instruction is to be executed, the user must restore the registers before executing the RTE. This restoration of registers is mandatory to properly restore the task that was interrupted. Upon a return through the user vector, address register A3 contains an address of a save area where the registers were saved. If A3 = 0, then no registers were saved (that is, no interrupt was taken and the RTE FLAG bit should be a 1).

Processor control is returned to the user in a variety of ways:

intermediate status:

WAITTIME	\$00	WAIT TIME
WAITINTC	\$01	WAIT FOR INTERRUPT (CLOSED)
WAITINTO	\$02	WAIT FOR INTERRUPT (OPEN)
LINKIRQ	\$03	LINK FLAG
MSGRCVD	\$04	MESSAGE RECEIVED
LONGMSG	\$05	ALLOCATE MESSAGE BUFFER,

or final status.

(Refer back to Figure 10 and Table 3, and forward to paragraph E.9.4.2 for details.)

For the intermediate statuses, control is given back to the firmware in two ways. One is through an NCR 5380 interrupt (WAIT FOR INTERRUPT (OPEN), or WAIT FOR INTERRUPT (CLOSED)). The other return mechanism is through a direct branch or jump into the REACTIVATION entry point of the SCSI firmware. The WAIT FOR INTERRUPT (CLOSED) status is usually given when a particular target is "threaded" to the MVME117 on the SCSI bus, and is slow in transitioning between information transfer phases. The processor is returned to the user in the WAIT FOR INTERRUPT (CLOSED) case, but the bus is occupied, so no new commands may be issued. A bus phase mismatch interrupt brings the processor back to the SCSI firmware to finish the command execution that was temporarily slowed down by the target. The other interrupt intermediate status is the WAIT FOR INTERRUPT (OPEN). This status is passed to the user when a target disconnects from the bus, and resumes with the reselection of the MVME117. For the WAIT FOR INTERRUPT (OPEN) status, the user may send a new command because the firmware temporarily is idle.

The second method of returning control involves the direct branch or jump to the REACTIVATION entry point of the firmware. For all the statuses involved (WAIT TIME, LINK FLAG, MESSAGE RECEIVED, ALLOCATE MESSAGE BUFFER, or final status with COMMAND PENDING bit set), the MVME117 is the current SCSI bus initiator and the user may only service the current "thread".

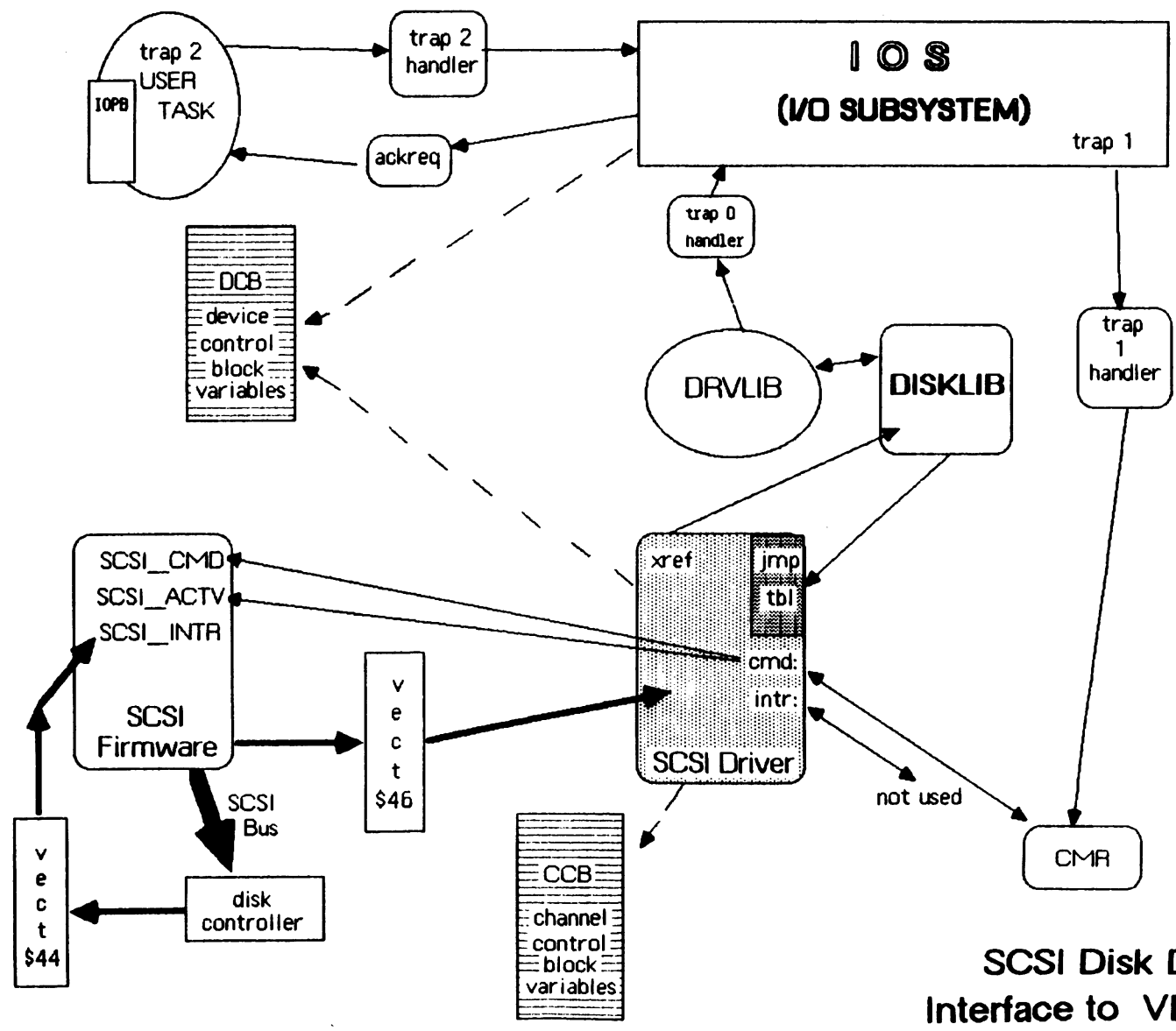
The MVME117 SCSI firmware was designed to operate in both interrupt and non-interrupt modes. When the user packets choose the interrupt mode of operation, the NCR 5380 interrupt is enabled at level 2 in the MVME117 interrupt handler. Vector number \$44 is used by the NCR 5380 for the SCSI bus interrupts and the SCSI firmware initializes vector offset \$110 to point to the SCSI firmware interrupt handler.

Whenever processor control is passed to the SCSI firmware interrupt handler, the MC68010 interrupt mask is at level 2. As processor control is switched out of the interrupt handler, the MC68010 interrupt mask is still at level 2. The device interface routines lower the interrupt mask to level 1 whenever there is a chance that an interrupt may catch an abnormal or transitional event such as a bus parity error or a false disconnect.

The user has the option of using fast pseudo-DMA in a custom SCSI sequence by setting the DMA on bit to zero in the Status/Flag byte of the Command Table (see Figure 9). If the user utilizes the fast pseudo-DMA channel, and if the target does not maintain a continuous fast data rate, a bus error may result. Whenever fast pseudo-DMA is used, the bus error handler of the SCSI firmware is invoked. If a bus error occurs during the pseudo-DMA operation, the SCSI firmware bus error handler will be executed. This bus error handler raises the interrupt mask to level 7 for a duration of seven instructions, then lowers the mask to level 1 and resumes execution of the SCSI function that was being executed.

## E.9 WRITING A DRIVER

This section covers information essential in writing a driver to support the SCSI interface. The approach taken is to describe the major sections of a driver that need to be written. The examples shown have been extracted from the VERSAdos SCSI driver, and are dependent on the driver interface to the VERSAdos operating system. See Figure 12 for this interface, and Figure 13 for details of the interaction between the driver and the SCSI firmware.



SCSI Disk Driver Interface to VERSAdos

FIGURE 12. SCSI Disk Driver Interface to VERSAdos



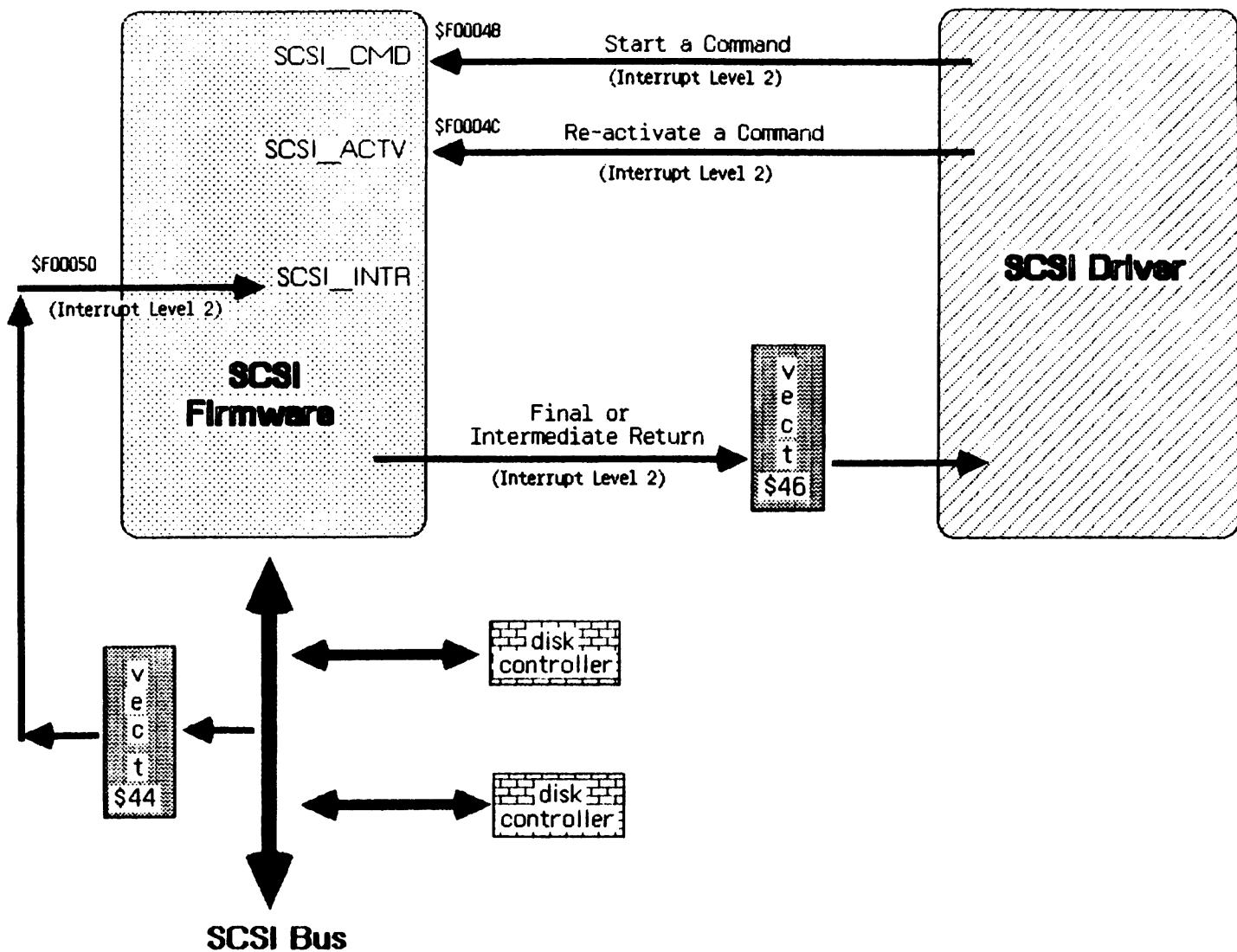


FIGURE 13. SCSI Disk Driver

### E.9.1 Introduction

Any driver that communicates to the SCSI firmware starts by building a command packet in memory and calling the command entry point (SCSI\_CMD) in the MVME117bug. The address of the packet is contained in register A2. If the SCSI firmware cannot complete the command, it expects to be reactivated by the driver calling the command reactivation routine (SCSI\_ACTV) in the MVME117bug.

Access to the three SCSI entry points is provided through the use of a "Branch Table" located within the beginning of the MVME117 debug monitor. The branch table entry points and their SCSI firmware functions are shown in the following list.

SCSI_CMD	EQU	\$F00048	SCSI command entry.
SCSI_ACTV	EQU	\$F0004C	SCSI command reactivation entry.
SCSI_INT	EQU	\$F00050	SCSI interrupt entry.

Interrupts from the SCSI controller chip are through vector \$44 (offset \$110). The SCSI firmware sets this vector to point to its interrupt entry point.

Return from the SCSI firmware to the driver is done through the vector supplied in the packet. The VERSAdos driver uses vector \$46 (offset \$118). This vector must be initialized to point to a driver routine which handles the return. Refer to paragraph E.9.4 for details on the return.

Commands coming from the operating system must be queued in the SCSI driver. The SCSI firmware can accept only one command per device. A busy error is returned if more than one command is received for a device. Until that command has been completed, or abnormally terminated, no other command may be dispatched to the same device. Commands to any other device may be issued after a "Wait for Interrupt Open" (\$02) intermediate status, or final status is returned from the SCSI firmware to the driver.

### E.9.2 Building the Packet

Paragraph E.3 provides the details necessary to build the command packet. After the packet has been built, it needs to be passed to the SCSI firmware.

### E.9.3 Passing Commands to the SCSI Firmware

Actual passing control from the driver to the SCSI firmware can be done in several ways.

- a. Jumping directly to the firmware from the driver. The return from the firmware back to the driver is through the vector supplied within the packet.
- b. Schedule an activation call in the driver. Control can be returned to the system. When the driver is activated (at interrupt level 2), control can then be passed to the firmware. Return to the driver by the firmware is through the vector supplied within the packet. At this point the activation call can be completed, returning control

back to the system. This is the approach provided within the VERSAdos operating system. The VERSAdos driver makes a timer call. Once the timer is running, the driver returns control back to the operating system. When the driver is re-entered (by the timer), it jumps to the SCSI command entry (SCSI\_CMD) of the firmware. This timer routine runs at the hardware interrupt level (2).

The VERSAdos driver could simply jump to the firmware command entry point (if the firmware is not busy) but instead uses a timer call. Once the timer has been set up, the driver exits and returns control back to the system. This allows other tasks to run. It also simplifies the work that needs to be done when the SCSI firmware returns control back to the driver. If the SCSI RTE bit is active on the return, then an interrupt from a disk controller must have caused the firmware to run. In this case the driver needs to do an RTE. If it is not a SCSI RTE, then it must have been the driver's timer call coming in which caused the firmware to run. In this case a simple RTE cleans up after the driver's call.

**E**

#### E.9.4 Return to the Driver from SCSI Firmware.

##### NOTE

The return to the driver by the SCSI firmware is through the vector supplied in the packet. Register A2 is set pointing to the packet.

Depending on the status set in the packet, the packet may not be valid for a command. This can occur if the SCSI firmware receives a command or an interrupt when it is not expected. If the SCSI firmware receives a "SCSI bus" reset from some other device on the SCSI bus, then the firmware simply returns to the driver with the status set, allowing the driver to handle the problem. Register A2 will be pointing to the packet that was used to request the last Attach.

The first step upon returning is to check for Final or Intermediate status.

The code used in the VERSAdos driver is:

FIN_STAT	EQU	15	Bit 15 = 0 = Final Status
S_STAT	EQU	2	SCSI Status word (2nd word in packet)

```
BTST.W  #FIN_STAT,S_STAT(A2) Check for final/intermediate status.
                                     (A2 is pointing to the packet)
BEQ.S   FINAL_STATUS          FINAL STATUS...
```

**E.9.4.1 Final Status.** If the status is Final status, then the VERSAdos driver will:

- a. Complete the command by posting status back to the operating system.

b. See if the firmware has done preprocessing for a command.

1. Check if SCSI COMMAND PENDING flag is set (see Figure 10).

After a SCSI target disconnects from the MVME117, the firmware returns the processor to the driver with an intermediate status of "Wait for Interrupt (Open)" code \$02. This status indicates that a Reselection Interrupt will bring the processor to the firmware. A new command for a different device may be sent to the firmware as a result of the \$02 intermediate status. Only if a new command is sent to the SCSI firmware, does a phenomenon called "Preprocessing" occur. This preprocessing refers to the table manipulation and command preparation that is performed by the user interface (USERINT) module of the SCSI firmware. Every command goes through this preprocessing. The phenomenon occurs when a preprocessed command is slower to acquire the SCSI bus than a target is to reselect the MVME117 by acquisition of the bus. The sequence that takes place is as follows:

- (a) A target reselects the MVME117 and re-establishes the thread that was earlier disconnected. This causes the preprocessed command to be saved in the firmware preprocessing queue.
- (b) When this command is completed and the threaded target disconnects from the bus, the SCSI firmware returns to the driver with final status for that command.
- (c) At final status the "COMMAND PENDING bit" in the status word for the command indicates that a preprocessed command is in the SCSI firmware queue.
- (d) The driver needs to pass control to the SCSI firmware command reactivation entry (SCSI\_ACTV). The VERSAdos driver sets up a timer call so that it can REACTIVATE the firmware later. The reason is the driver must complete processing for the current command before re-entering the SCSI firmware.
- (e) The preprocessed command is then removed from the queue and threaded to the selected SCSI device.

2. Not preprocessing.

See if there is something in the driver queue to work on. If there is, then set up a timer call where the command is given to the firmware. After the driver sets up the timer call, it exits back to the system. The driver can't simply jump to the SCSI command handler because the last command that it sent to the firmware needs to be finished. When the driver gets control from the timer, it jumps to the firmware reactivation entry point (SCSI\_ACTV).

c. Common exit point for SCSI\_RETURN (Final or Intermediate status):

Check if the SCSI firmware took an interrupt. If it did we need to:

1. Restore the registers the firmware used (pointed to by A3).  
Registers are restored as follows:

RESTORE REGISTERS FOR SCSI ROUTINE

The driver must restore the registers the SCSI firmware has used. If the registers don't need to be restored, the firmware sets A3 to zero. Otherwise, A3 is pointing to a list of the registers to restore.

Entry:     A3   = 0     No need to restore registers.  
              <> 0     Pointer to register list of registers to  
                          restore.

Exit:       back to caller.

RESTORE\_REG:

```

                                CMP.L     #0,A3
                                BEQ       REST_OUT

                                MOVE.L    #14,D0           Number of longwords to move.
                                ADDA.L    #60,A3           Start from bottom of register
                                                                list.
RSTRLOOP   MOVE.L   -(A3),-(SP)       Store on the stack for MOVEM
                                                                instruction.
                                DBRA     D0,RSTRLOOP

                                MOVEM.L   (SP)+,D0-D7/A0-A6   Restore registers.

REST_OUT   RTS

```

2. Clean up the stack with an RTE.

If it is not a return from the SCSI interrupt, then the VERSAdos driver cleans up after the timer call. In that case, the driver needs to:

- (a) Restore those registers that the RMS68K timer handler does not save.
- (b) Do an RTE to finish the timer call.

**E.9.4.2 Intermediate Status.** The operations that need to be done for the Intermediate status of WAIT TIME, WAIT FOR INTERRUPT CLOSED, WAIT FOR INTERRUPT OPEN are:

- a. WAIT TIME: Status code (byte 3) = \$00

The firmware needs a delay of a specified amount of time. Register A1 contains the number of milliseconds to delay. The VERSAdos driver makes a timer call to do the delay. Once the timer is running, it exits back to the system through the common exit code. (Refer to paragraph E.9.4.1.c.) When the delay has expired and the driver regains control from the timer, the firmware is reactivated by jumping to the command reactivation entry (SCSI\_ACTV).

#### NOTE

Typically the delay requested is for 250 ms. This specifies the minimum time to delay before reactivating the firmware.

- b. WAIT FOR INTERRUPT (CLOSED): Status code (byte 3) = \$01

SCSI firmware is returning the processor to the driver. The firmware is unable to take any more commands at this time. The firmware returns again when it is OK to send a command (with WAIT FOR INTERRUPT (OPEN) intermediate status, or final status), or when the driver needs to reactivate the firmware for the preprocessed case (at final status). The VERSAdos driver then exits back to the system through the common exit code. (Refer to paragraph E.9.4.1.c.)

- c. WAIT FOR INTERRUPT (OPEN): Status code (byte 3) = \$02

SCSI firmware is returning the processor to the driver. The firmware may be able to take more commands at this time. If there is a command in the driver queue, then the VERSAdos driver sets up a timer call where a new command can be given to the firmware. In any case, it exits back to the system through the common exit. (Refer to paragraph E.9.4.1.c.)

#### CAUTION

THE VERSAdos DRIVER HANDLES ONLY WAIT TIME, WAIT FOR INTERRUPT (CLOSED), AND WAIT FOR INTERRUPT (OPEN). ANY OTHER INTERMEDIATE CODES CAUSE THE CHANNEL TO BE MARKED DOWN! AFTER THE DRIVER MARKS THE CHANNEL AS DOWN, IT EXITS BACK TO THE SYSTEM THROUGH THE COMMON EXIT. (REFER TO PARAGRAPH E.9.4.1.c.)

**E**

The other intermediate status returns are:

- d. LINK FLAG: Status code (byte 3) = \$03 (custom SCSI sequence packets only)

The firmware is returning control back to the driver. A Linked command has completed one portion of a "Linked Command with Flag". The user may consider this return the "Link Flag Interrupt". Control is returned to the vector supplied within the packet. Enter the SCSI firmware at the Reactivate entry (SCSI\_ACTV). Refer to the SCSI bus draft for more details. The link flag status applies only to custom SCSI sequence packets using the linked command capability.

- e. MESSAGE RECEIVED: Status code (byte 3) = \$04

The firmware is returning control back to the driver through the vector supplied within the packet. An extended message has been received that the SCSI firmware can not interpret. The user is allowed to interpret this message, then return control to the SCSI firmware at the Reactivate entry (SCSI\_ACTV).

The MESSAGE RECEIVED intermediate status is passed to the driver after it has provided the buffer area to hold the extended message and the extended message has been placed in that buffer. The firmware does not acknowledge the sender on the last byte of the message until reactivated by the driver. The driver can reject the message by adjusting the script pointer to point to a new script.

- f. ALLOCATE MESSAGE BUFFER: Status code (byte 3) = \$05

The firmware is returning control back to the driver through the vector supplied within the packet. A SCSI target has sent an extended message to the MVME117 and the driver has not allocated enough buffer space to store the message. The driver is now requested to point the Message-in pointer within the command table to an area big enough to hold the message. (The longest possible message is 258 bytes.) Control is then returned to the SCSI firmware at the Reactivate entry (SCSI\_ACTV).

#### NOTE

The ALLOCATE MESSAGE BUFFER intermediate status is received only when another device sends an extended message to this MVME117 module.

**E**

## INDEX

- AB (enable autoboot) 2, 10, 17, 24, 26-28, 32, 43, 58, 94, 112 (see also autoboot)
- ABORT 2, 9, 27, 38, 54, 98, 143, 147
- ABORT switch 2, 9, 38, 98, 143, 147
- absolute address(es) 21, 44, 47, 50, 54, 120, 124, 125
- address(es) 20-22
- algorithm 45
- alignment 81
- allocate 158, 183, 187, 188, 196
- ANSI 5
- arithmetic logic unit 17, 95
- ASCII 10, 14, 15, 19, 26, 35, 40, 41, 52, 75, 81, 82, 92, 94, 97, 101, 113, 119-122, 132, 134, 135, 154, 155
- assembler(s) 20, 79, 115-117, 119-121, 124-126, 128, 152, 153
- assembly 79, 124, 126, 128, 129
- assembly language 115, 118, 121, 154
- assign alternate 60, 132, 136, 159, 160, 166, 168-170
- attach 24, 43, 60, 61, 69, 85, 93, 100, 113, 132, 136, 139, 141, 160, 162, 164, 165, 183, 192
- Autoboot 2, 10, 11, 17, 24, 26-30, 32, 35, 37, 43, 58, 94, 112, 147 (see also AB and NOAB)
- baud rate 87-89
- BD (boot dump) 4, 7, 24, 28-30, 58, 112
- BERR 183 (see also bus error)
- BF (block fill) 23, 28, 31, 58, 62, 73, 82, 109, 112 (see also block fill)
- BH (boot halt) 4, 5, 24, 28, 32, 58, 112
- BI (block initialize) 23, 28, 33, 42, 58, 112
- binary 31, 48, 120, 127, 151, 154, 155
- block fill 31, 112 (see also BF)
- block move 34, 112 (see also BM)
- BM (block move) 23, 28, 34, 58, 112 (see also block move)
- BO (boot load) 4, 5, 17, 24, 28, 32, 35-37, 58, 112
- bootstrap 29
- boundary(ies) 14, 15, 31, 33, 38, 42, 119, 147
- BR (set breakpoints) 19, 21, 24, 28, 38, 39, 53, 56-58, 83, 91, 107, 112 (see also breakpoints)
- branch instructions 48, 119
- breakpoint(s) 2, 19, 22, 24, 38, 39, 49, 53-57, 90, 91, 105, 107, 112, 113, 145, 147 (see also BR and NOBR)
- BS (block search) 19, 23, 28, 40, 41, 58, 81, 112
- BT (block test) 23, 28, 33, 42, 58, 112, 118, 119

buffer	81, 82, 102, 132, 158, 172, 180, 183, 184, 187, 188, 196
bus arbitration	180
bus error	2, 3, 7, 10, 12, 15, 33, 95, 111, 126, 127, 188 (see also BERR)
byte address	151-154
 CDB (command descriptor block)	 177-180
channel	4, 5, 51, 149, 180, 188, 195
character(s)	3, 12, 19, 22, 29, 33, 35, 37, 51, 79, 81, 87-89, 96, 97, 101-104, 108, 113, 114, 116, 117, 119-122, 125-128, 132, 145, 151, 154, 155
checksum	10, 14-16, 18, 24, 44-47, 72, 95, 108, 112, 151, 152, 154, 155 (see also CS)
clock	94, 97, 99, 101, 146
CMOS (display CMOS RAM variables)	2, 10, 11, 14, 17, 24, 26-28, 43, 58, 92-94, 96, 99, 112, 146
cold start	2, 7, 11, 99, 100, 111, 113, 141, 147
compatibility	151
complement	151
computer	6, 12, 74, 109, 151
condition code	120
conditional assembly	116
configuration	4-7, 10, 29, 36, 68, 69, 71, 95, 97, 100, 113, 140, 141, 149, 150
console	3, 10, 12, 22, 27, 33, 51, 72, 75, 85, 86, 96, 101, 108, 132, 133, 143
control blocks	76, 137
control characters	12, 51
control functions	12
control register	95, 98, 141, 186
controller	4-6, 10, 11, 14, 17, 24, 26-30, 32, 35-37, 43, 58-68, 70, 71, 93, 95, 97, 99, 112, 113, 132, 137, 140, 141, 145, 159-167, 169-171, 174-178, 180, 183-185, 191
conversion	4, 23, 48, 97, 131
coprocessor	131
CPU	17, 25, 95, 96
CRT	51, 74, 123
CS (checksum)	14-16, 24, 28, 44-47, 58, 112 (see also checksum)
custom	60, 132, 137, 160, 162, 171, 173, 174, 177, 178, 183, 188, 196
cycle	157
 data block	 165
data register(s)	23, 25, 114, 119, 120, 143
date	43, 94, 101
DC (data conversion)	23, 28, 31, 48, 58, 112

debug	1, 2, 4, 7, 9-11, 17, 18, 27, 32, 43, 94, 95, 100-103, 115, 116, 124, 131, 140, 157, 159, 191
defect list	60, 132, 136, 160, 166-169
detach	24, 60, 85, 132, 136, 139, 160, 164
DF (display formatted registers)	23, 25, 28, 49, 50, 56, 58, 83, 91, 105, 107, 112, 143
diagnostic(s)	7, 65, 92, 97, 146
diagnostic test	95
direct access	164
disassembler	31, 52, 75, 76, 80, 118, 121, 123, 126
disk access	7
disk controller(s)	2, 159, 191
diskette	150
DMA (direct memory access)	172, 179, 180, 183, 186, 188
driver	149, 171, 188-196
DU (dump memory)	24, 28, 51, 52, 58, 73, 74, 85, 109, 110, 112
dumb terminal	102
ECC (error correction code)	60, 149, 185
echo	72, 74, 108, 133
ENTRY	153
entry point(s)	36, 95, 140, 153, 157-159, 187, 188, 191, 193
EPROM	1, 7, 14, 131, 140
error correction code	60
error message(s)	3, 5, 22, 35, 61, 64, 116, 124, 126, 145, 146
exit(s)	79, 87, 94, 101-105, 113, 122-124, 128, 191, 193-195
external interrupt	187
failure(s)	10, 26, 33, 44, 95, 99, 146, 184
fault	3, 17, 185
flag(s)	36, 158, 162, 175, 179-183, 187, 188, 193, 196
floating point	131
floppy disk drive	68
frame	3, 131
G (execute program)	9, 24, 28, 53-55, 58, 112 (see also G0)
GD (go direct)	24, 28, 53, 56, 58, 112
G0 (execute program)	9, 28, 38, 53-56, 58, 91, 105, 107 (see also G)
GT (go until breakpoint)	22, 24, 28, 38, 39, 53, 56-58, 105, 107, 112, 145

halt	24, 32, 112
handshaking	131
hard copy	94, 101, 125
HE (help)	17, 24, 27, 28, 32, 58, 112
header	15, 63, 153
header record	152-154
help	17, 24, 27, 32, 58
I/O devices	79
IDENT	152
ignore	72, 108
illegal instruction(s)	2, 7, 38, 140, 145
index registers	119
indirect address	55
initial program loader	36
initialization	7, 12, 14, 15, 27, 111
input area	133
input routine	12, 19
instruction set	117
instructions	17, 31, 90, 91, 95, 105, 115, 117, 118, 121, 123, 124, 126, 188
interactive	87, 94, 115
interrupt handler	133-135, 158, 183, 188
interrupt mask	188
interrupt vector	143
IOC (I/O command)	4, 24, 28, 58-65, 112
IOP (I/O physical)	4, 7, 24, 28, 58, 60, 63-67, 113
IOT (I/O teach)	4, 5, 24, 28, 58, 68-71, 113
IPL (initial program loader)	36, 37
IRQ (interrupt request)	183
jump	124, 171, 187, 188, 191, 193
jumpers	89
label(s)	79, 115-117, 121, 122, 124
LED	17, 95, 98, 99, 146
line printer	85
LINK FLAG	158, 187, 188, 196
linkage	14, 131, 140
linkage editor	153
linker(s)	152, 153
LO (load)	19, 24, 28, 51, 58, 72-74, 109, 113
load module	153
logic circuits	7
logical unit number	161, 163, 167, 170, 174, 176 (see also LUN)
longword(s)	14, 17, 40, 78, 79, 81, 117-119, 169, 194
loopback test	96

LUN 61, 62, 161, 163, 167, 170-174, 176, 178, 183 (see also logical unit number)

M (memory modify) 16, 23, 28, 46, 47, 55, 58, 61, 64, 67, 78-80, 90, 103, 104, 109, 110, 113, 127 (see also MM)

machine instructions 115

machine language 115

macro 116

mark 59, 65, 69, 88, 116, 182, 185

mask 40, 41, 112, 149, 188

mass storage 69

MCR 97, 141 (see also module control register)

MD (memory display) 15, 16, 22, 23, 26, 28, 31, 34, 40, 43, 46, 52, 58, 62, 66, 73-77, 81-86, 92, 94, 105, 109, 113, 123, 125, 126

memory address(es) 3, 35, 62-64, 66, 67, 75, 121, 128, 140, 141, 151, 161

memory dump 147

memory fill 23

memory map 1, 10, 11, 14

menu 71, 87, 88

message(s) 3, 7, 10-12, 17, 22, 24, 26, 27, 33, 63, 85, 87, 93, 95, 96, 98, 99, 105, 108, 111, 116, 126, 133, 134, 145-147, 157, 158, 172, 175, 177-180, 182, 183, 187, 188, 196

MM (memory modify) 16, 28, 58, 59, 78-80, 84, 94, 102, 103, 109, 110, 123, 124, 127, 128, (see also M)

module control register 141 (see also MCR)

monitor(s) (MONITOR) 1-5, 7, 9-12, 14, 15, 17, 18, 20-24, 27, 29, 31, 32, 35-38, 42-44, 46, 49, 79, 81, 85, 87, 94, 95, 98-100, 102, 103, 105, 108, 111-116, 124, 131-133, 140, 143, 145, 147, 149, 150, 157, 159, 191

MPU (microprocessor unit) 10, 17, 95, 99, 135, 146

MS (memory set) 19, 23, 28, 58, 81, 82, 113

multitasking 157

MVME050 2, 6, 28, 51, 58, 72, 75, 85, 86, 89, 90, 100, 102, 103, 109, 133, 135

MVME316 4, 5

MVME400 2, 6, 28, 51, 58, 72, 75, 89, 90, 100, 102, 103, 109, 135

MVME410 2, 6, 28, 51, 58, 75, 85, 90, 103, 134, 135

MVME708-1 6, 51, 72, 75, 85, 100, 102, 109, 135

NOAB (disable autoboot)  
 NOBR (remove breakpoints)  
 NOPA (detach printer)  
 NOPT (disable power-up test)  
 nulls  
  
 object code  
 OF (display offsets)  
 offset  
  
 offset register  
 opcode(s)  
 operand(s)  
 options  
  
 OR  
 output modules  
 overflow  
 overwrite  
  
 PA (printer attach)  
  
 packet(s)  
  
 parameter(s)  
  
 parity  
 parity error  
 PC  
  
 PF (port format)  
 phase(s)  
 pointer(s)  
 polling  
  
 24, 26-28, 37, 58 (see also autoboot)  
 19, 24, 28, 38, 39, 58 (see also breakpoints)  
 24, 28, 58, 85, 86 (see also printer attach)  
 24, 28, 58, 92 (see also power-up)  
 87-89  
  
 75  
 1, 21, 23, 25, 28, 49, 58, 65, 83, 84, 96, 113, 153, 157 (see also offset)  
 3, 7, 14, 15, 20-22, 25, 47, 48, 51, 72-74, 83, 84, 102, 140, 145, 149, 181-183, 185, 188, 191 (see also OF)  
 15, 21, 23, 25, 44, 47, 49, 50, 54, 55, 61, 62, 72, 73, 83, 113, 114  
 118, 119, 128, 129, 145, 154  
 16, 32, 44, 46, 47, 54, 56, 81, 103, 115-119, 125, 127, 145  
 19, 20, 39, 40, 59, 65, 68, 69, 72, 75, 76, 78, 87, 90, 103, 105, 108, 112, 113, 145, 157  
 18, 34, 129  
 151  
 182, 184, 185  
 123  
  
 24, 28, 58, 82, 85, 86, 94, 100, 101, 113, 123, 125 (see also printer attach)  
 43, 59-64, 67, 69, 93, 131, 132, 135-137, 139, 141, 157, 158, 160-167, 169-171, 173-176, 178, 180-185, 187, 188, 191, 192, 196  
 5, 20-22, 29, 30, 36-38, 44, 54, 59, 65, 68-71, 81, 83, 87, 88, 97, 132, 141, 147, 149, 150, 159, 162, 184  
 17, 18, 33, 87, 89, 95, 96, 99, 146, 157, 172, 179, 180  
 95, 183, 184, 186, 188  
 22, 23, 25, 28, 32, 49, 50, 53, 55-58, 63, 67, 90, 91, 93, 105-107, 114, 120, 125, 127, 128, 147 (see also program counter)  
 24, 28, 39, 51, 58, 87-91, 102, 103, 105, 113  
 93, 157, 172, 177, 178, 180, 182, 183, 185-187  
 4, 105, 119, 162, 167, 171-175, 177-181, 194, 196  
 157, 186

port(s)	2, 6, 10, 12, 17, 20, 24, 51, 52, 58, 72, 75, 85-87, 89, 90, 96, 99, 100, 102, 103, 108, 109, 112-114, 123, 125, 132-135, 146, 157 (see also PF)
power-up	10, 11, 14, 15, 17, 21, 24, 26-28, 43, 44, 46, 58, 92, 95, 113, 147, 162 (see also NOPT and PT)
printer attach	24, 72, 82, 85, 94, 101, 125 (see also NOPA and PA)
privileged instructions	120
program counter	2, 3, 7, 23, 25, 36, 48-50, 53-55, 105, 114, 119, 120, 143 (see also PC)
program registers	56
prompt(s)	9, 11, 20, 26, 27, 29, 35, 43, 59, 61, 64, 65, 68-70, 78, 79, 82, 87-89, 94, 101, 105, 107, 108, 127, 128, 147
protocol	93, 157, 183
PRT	28, 58, 90, 103 (see also port(s))
PT (enable power-up test)	10, 17, 24, 28, 58, 92, 113 (see also power-up)
PTM (programmable timer module)	97, 99, 146
queue	193, 195
RAM	2, 7, 10, 11, 14, 15, 17, 18, 24, 26, 27, 29-32, 35, 37, 40, 43, 54, 55, 60-62, 66, 67, 73, 87, 90, 92-96, 98, 99, 102, 103, 109, 111, 112, 121, 133, 140, 141, 143, 146, 147, 162-164, 180, 185
read cycles	7
real time	99, 146
real-time clock	97
records	73, 74, 152-154
recoverable error	184
reset (RESET (bus or controller reset))	3, 7, 10, 11, 19, 21, 24, 27-29, 38, 54, 58, 60, 73, 83, 93, 95, 97, 111, 113, 132, 137, 141, 143, 160, 162, 175-177, 183, 184, 186, 191
reset switch (RESET switch)	92, 143
ROM	7, 10, 12, 14, 18, 44, 54, 95, 99, 126, 140, 141, 145, 146
ROMBOOT	10, 14, 15, 44
RS-232C	6, 10, 52
RWIN	2, 5, 28, 37, 58-60, 63, 64, 67, 70, 71, 112, 113, 150 (see also winchester disk)

- S-record(s) 24, 51, 72-74, 108, 110, 112, 113, 151-155
- SASI 60, 66, 132, 136, 157, 160, 162, 164, 166, 169, 170, 180, 183
- script 171-174, 178, 181, 183, 196
- SCSI 2, 4-6, 24, 26, 28, 37, 43, 58-62, 65-70, 93, 97, 99, 113, 131, 132, 135-139, 141, 146, 157-181, 183, 185-196 (see also small computer system(s) interface)
- SCSI bus 5, 60, 93, 131, 132, 137, 157-160, 175, 177-180, 183, 186-188, 191, 193, 196
- sector 4, 5, 29, 35, 36, 60-62, 68, 70, 71, 97, 99, 132, 135, 136, 149, 150, 159-161, 163-170, 172, 185
- semaphore 4, 7, 141
- SET (set date/time) 23, 24, 28, 43, 58, 94, 101, 113
- set breakpoint 19 (see also BR and breakpoints)
- sign 1, 20, 40, 72, 108, 119, 120, 127
- signal 185, 186
- small computer system(s) interface 4, 6, 157 (see also SCSI)
- software abort 2, 7, 9, 10, 27, 140, 143, 147
- source code 121, 128
- source statement 75, 116, 117
- ST (self-test) 5, 24, 28, 58, 92, 95-99, 113
- stack 2, 3, 15, 17, 18, 32, 36, 105, 119, 127, 131, 132, 135-137, 140, 143, 147, 194
- stack pointer 2, 7, 23, 25, 36, 49, 50, 114, 119, 120, 143
- status bit 162
- status byte 131, 135, 161, 163, 165, 167, 170, 173, 176
- status register 3, 7, 23, 25, 36, 49, 50, 114, 119, 120, 131, 135, 143
- string 14, 15, 29, 35-37, 40, 41, 81, 82, 94, 97, 112, 119, 127, 132, 134, 135
- subroutine(s) 36, 140
- summary 112-114, 121, 132
- supervisor 2, 23, 25, 36, 49, 50, 114, 120, 143, 157
- symbol 19, 59, 65, 69, 88, 116
- SYSFAIL (system fail) 11, 14
- system controller 6, 102
  
- T (trace) 24, 28, 58, 90, 91, 105, 106, 113 (see also TR and trace)
- TA (terminal attach) 24, 28, 58, 100, 113
- task 15, 117, 187

terminal	1, 3, 10, 12, 22, 24, 40, 51, 52, 56, 72, 75, 76, 81, 85, 86, 94, 95, 100-102, 108, 109, 111, 113, 123, 125, 132, 135, 138
TIME (display date/time)	24, 28, 58, 94, 101, 113, 158, 185, 187, 188, 194, 195
time-of-day clock	94, 101
time-sharing	152
TM (transparent mode)	24, 28, 58, 81, 87, 102-104, 113, 114
TR (trace)	28, 56, 58, 105-107 (see also T and trace)
trace	2, 24, 38, 90, 91, 105, 107, 113, 140, 147 (see also T and TR)
track	60, 61, 63, 64, 68, 132, 136, 149, 150, 159, 160, 163, 164, 166, 169, 170, 185
transition	6, 51, 72, 75, 85, 100, 102, 109, 135, 183, 186
trap	1-3, 7, 12, 33, 36, 62, 105, 126, 127, 131-138, 140, 145 (see also TRAP #15)
TRAP #15	1, 2, 7, 36, 62, 131-138, 140 (see also trap)
TT (trace to temporary breakpoint)	24, 28, 39, 56, 58, 105, 107, 113
update	78, 79
utilities	153
VBR	3, 4, 22, 23, 25, 28, 32, 49, 50, 53, 57, 58, 63, 67, 90, 91, 93, 106, 107, 114, 120, 127 (see also vector base register)
VE (verify)	19, 24, 28, 51, 58, 108-110, 113
vector(s)	2, 3, 7, 10, 12, 23, 25, 53, 61, 62, 111, 113, 114, 119, 120, 140, 141, 143, 145, 147, 158, 161, 163, 164, 167, 170, 171, 173, 174, 176, 187, 188, 191, 196 (see also VI)
vector base register	3, 23, 25, 114, 119, 120, 143 (see also VBR)
VERSAdos	4, 6, 27, 36, 37, 69, 97, 102, 149, 152, 153, 188, 189, 191-195
VI (vector initialize)	23, 28, 58, 100, 111, 113, 147 (see also vectors)
VMEbus	4, 7, 11, 141
VMEmodule	6, 7
warm start	3, 7, 11, 95, 111, 141, 147
winchester disk	4-6, 59, 63, 145 (see also RWIN)
WRITE	149, 185



**MICROSYSTEMS**

QUALITY • PEOPLE • PERFORMANCE

To: Motorola Inc.  
Microsystems  
2900 S. Diablo Way  
Tempe, Arizona 85282  
Attention: Publications Manager  
Maildrop DW164

[illegible]

Name \_\_\_\_\_ Title \_\_\_\_\_

Company \_\_\_\_\_ Division \_\_\_\_\_

Street \_\_\_\_\_ Mail Drop \_\_\_\_\_ Phone \_\_\_\_\_

City \_\_\_\_\_ State \_\_\_\_\_ Zip \_\_\_\_\_

**Motorola Computer Systems/Customer Support**  
(800) 528-1908  
(602) 438-3100

**MOTOROLA**



**MOTOROLA Semiconductor Products Inc.**

P.O. BOX 20912 • PHOENIX, ARIZONA 85036 • A SUBSIDIARY OF MOTOROLA INC.